BENCHMARKING OPTIMIZERS TO SOLVE INVERSE PROBLEMS WITH DIFFERENTIABLE PHYSICS SIMULATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Solving inverse problems with differentiable physics simulators holds the potential to revolutionize scientific discovery and engineering design, as it enjoys both the strict physical correctness from rigorous numerical physics simulators, and the high efficiency and effectiveness from automatic differentiation and gradient-based optimization. However, currently, this paradigm faces performance issues in optimization. In this work, we target benchmarking the performance of different optimizers to solve various inverse problems. We construct 12 differentiable physics simulators spanning physics domains including discrete mechanics, continuous mechanics, atomistic simulations, rendering, and semi-empirical physics models. Based on these simulators, we design corresponding inverse problems that can be categorized into parameter identification, inverse design, and optimal control. Finally, we conduct extensive experiments to compare the performance of different optimizers, including regular first-order methods, approximate second-order methods, as well as global optimizers, on these inverse problems, and analyze the results to provide insights on how to choose and design optimizers for differentiable programming. We hope such benchmarks can inspire the development of more effective optimizers, and further promote the applications of differentiable programming in various scientific and engineering domains.

1 INTRODUCTION

Deep learning has achieved remarkable success in the last decade, relying on the universal approximation capability of neural network models (Scarselli and Tsoi, 1998), the high efficiency of gradient computation of automatic differentiation systems (Baydin et al., 2018), and the incredible effectiveness and generalizability from the optimization process (Chizat and Bach, 2020). However, purely data-driven models can never solve the issue of unguaranteed validity. One of the alternatives is differentiable physics models, which guarantee strict physical rigorousness by implementing the numerical physics process in automatic differentiation systems, while still enjoys the high efficiency and effectiveness provided by automatic differentiation and gradient-based optimizers.

Currently, differentiable physics models are facing performance issues. Although it achieves great success in certain scenarios, in some others the optimization process tends to fail. As a matter of fact, optimization in deep learning has never been an easy task. During the development of deep learning, many model architecture designs are specifically proposed so that training can be more effective and stable, such as ReLU (Nair and Hinton, 2010), residual connections (He et al., 2016), normalizations (Huang et al., 2023), etc. However, such techniques can not be applied to differentiable physics models, because the model structure is predetermined by the underlying physical process. Hence this approach can only be improved by advancing optimizer techniques.

To the best of our knowledge, there are no comprehensive benchmarks available for evaluating the performance of optimizers on various differentiable physics models and inverse problems. Many previous works have implemented differentiable physics models for all kinds of physical domains and tasks in the last few years, but very few works have systematically studied this paradigm as a whole. It is difficult to unify these simulators and tasks into a framework, as each of them may use different coding structures, different automatic differentiation tools, and even different languages,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

which induces difficulties to discover better methods that can be widely applied in various scenarios. The closest previous works we can find are DiffTaichi (Hu et al., 2020) and PhiFlow (Holl and Thuerey, 2024). However, their motivation and target are neither to benchmark optimizers nor to systematically study the performances on various physics models and inverse problems.

In this work, we target benchmarking the performance of different optimizers to solve inverse problems with differentiable physics simulators. We construct 12 differentiable physics simulators spanning areas including discrete mechanics, continuous mechanics, atomistic simulations, rendering, and semi-empirical physics models. Correspondingly, we design inverse problems for each of the physics simulators, categorized into parameter identification, inverse design and optimal control problems. For optimizers, we utilize and implement first-order local, approximate second-order local, and global optimizers, each with 4 optimizers, all of them utilizing the gradient information to accelerate the optimization efficiency. Finally, we conduct extensive experiments and analyze the results. We hope such benchmarks can help make discoveries that are more fundamental and general about the optimization process of differentiable programming, inspire the development of more effective optimizers, and further promote the applications of differentiable programming in various scientific and engineering domains, as well as help general deep learning and large language model techniques to evolve to better model architectures and optimization methods.

2 Related Work

Differentiable Programming Techniques. Common deep learning layers such as fully-connected layers, convolution layers, and attention layers, are all explicit layers, which means the output of the layer z can be written as an explicit function of the input x, parameterized by trainable parameter θ, i.e., $z = f_\theta(x)$. Automatic differentiation, different from numerical differentiation or symbolic differentiation, can compute machine-precision gradients efficiently for models with high-dimensional input and low-dimensional output by backpropagation (Baydin et al., 2018).

For implicit layers, on the contrary, the input x and output z of the layer satisfy an equation $g_\theta(x, z) = 0$ (Kolter et al., 2020). In such cases, the gradient can be computed by the implicit differentiation method. Previous works deriving and implementing implicit layers have been proposed in various forms, including ODE (Ordinary Differential Equations) (Chen et al., 2018), PDE (Partial Differential Equations) (Mitusch et al., 2019; Rehmann et al., 2026), self-consistent iterations (Bai et al., 2019), optimization (Amos and Kolter, 2017; Agrawal et al., 2019), etc.

Differentiable Physics Models. Over the past decade, differentiable physics models have been developed across a broad spectrum of physical domains. In classical mechanics, differentiable formulations have been extensively advanced for both discrete and continuous systems. For discrete mechanics, researchers have successfully differentiated rigid-body dynamics (de Avila Belbute-Peres et al., 2018), mass-spring systems (Hu et al., 2020), and formalisms grounded in Lagrangian (Lutter et al., 2019) and Hamiltonian mechanics (Greydanus et al., 2019). Parallel progress in continuous mechanics has enabled end-to-end differentiable simulations of fluid mechanics (Holl and Thuerey, 2024), elastic mechanics (Hoyer et al., 2019), as well as wave propagation in electromagnetics (Oskooi et al., 2010), optics (Colburn and Majumdar, 2021), and acoustics (Jin et al., 2024). Beyond classical regimes, differentiability has been systematically integrated into ab-initio simulations, including DFT (Density Functional Theory) (Li et al., 2021; Kasim and Vinko, 2021), MD (Molecular Dynamics) (Doerr et al., 2021), and high-accuracy quantum chemical methods such as CCSD/CI (Tamayo-Mendoza et al., 2018). In computer graphics and vision, these techniques have further catalyzed advances in inverse rendering (Kato et al., 2020), such as NeRF (Neural Radiance Fields) (Mildenhall et al., 2021) and 3D Gaussian Splatting (Kerbl et al., 2023). Underpinning this rapid interdisciplinary expansion are dedicated programming ecosystems and frameworks explicitly engineered for differentiable programming, most notably JAX (Schoenholz and Cubuk, 2020; Freeman et al., 2021; Xue et al., 2023; Campagne et al., 2023; Du and He, 2025), Julia (Innes et al., 2019), and Taichi (Hu et al., 2020).

Optimizers for Deep Learning. Standard deep learning optimizers are predominantly first-order local optimizers with mini-batch gradient estimation, such as SGD (Stochastic Gradient Descent) and its variants with momentum and adaptive learning rate (Sun, 2020; Schneider et al., 2019). Recently, novel methods such as Muon (Jordan et al., 2024; Liu et al., 2025) have achieved substan-

tial improvement in training large language models, by explicitly leveraging the two-dimensional structure of neural network parameters. Concurrently, approximate second-order optimizers have been actively investigated for deep learning, including L-BFGS (Bollapragada et al., 2018), K-FAC (Martens and Grosse, 2015), Shampoo (Gupta et al., 2018), and SOAP (Vyas et al., 2025).

3 METHOD

In the following, we will introduce separately the differentiable physics models we implemented, the inverse problems we designed, and the optimizers we benchmarked in this work.

3.1 DIFFERENTIABLE PHYSICS MODEL ZOO

The physics models we implemented are summarized in Table 1. We consider 5 categories of physics simulations, i.e., discrete mechanics, continuous mechanics, atomistic simulations, rendering, and semi-empirical physics models, which can cover quite a portion of application scenarios such as academic research, industrial development, entertainment business, etc. All models are implemented in PyTorch, so that we enjoy a unified interface as well as the mature and active community support. All the physical and geometrical configurations, initial and boundary conditions, are provided in the `__init__` function. Only the control policies for controlled dynamic systems are fed from the `forward` function. The optimization variables are set as trainable, and the gradient of these variables with respect to the loss function is computed efficiently by the automatic differentiation system. We provide a gallery of the implemented differentiable physics simulators in Figure 1 to demonstrate the sanity of our implementation. More implementation details, such as the underlying mathematical formulations and numerical algorithms of each simulator, are provided in Appendix A.

Discrete Mechanics The discrete mechanics simulators consider systems made up of discrete components. Within this category, we consider a simple projectile motion problem which, from specific initial height, speed, launch angle, and gravity, computes flight time and horizontal range; a flight dynamics model simulating the flight trajectory with atmosphere, gravity, and controllable lift/drag coefficients; a 2D mass-spring robot with springs, damping, gravity, and inelastic ground contact, driven by a control policy; and a controllable 2D rigid-body dynamics system with damped springs and joints.

Continuous Mechanics The continuous mechanics simulators target macroscopic continuum problems, which are typically governed by PDEs. In the current benchmark, we implemented an elastic mechanics simulator and a fluid simulator, both in 2 dimensions. The elastic mechanics simulator (Christensen and Klarbring, 2008) computes the compliance and displacement of an elastic body under external forces and normals, given the material density and physical parameters. The fluid simulator computes the drag of a car silhouette in a wind tunnel (Angot et al., 1999).

Atomistic Simulations The atomistic simulations target microscopic problems at the atomic level. The DFT (Density Functional Theory) simulator (Bickelhaupt and Baerends, 2000) computes the total energy and electron density of a system of atoms, more specifically in our current implementation, a two-electron spherical closed shell. The TB (Tight Binding) model (Goringe et al., 1997) computes the supercell eigenvalue spectrum, Bloch bands, and (for visualization) a Gaussian-smoothed density of states, given onsite energies and hopping parameters.

Rendering The rendering simulations target image formation from environmental and rendering parameters. The Gaussian splatting simulator (Kerbl et al., 2023) renders the image by splatting the gaussian blobs with properties including position, scale, and color. The atmospheric scattering simulator (Rayleigh, 1871) generates the sky image by computing Rayleigh and Mie scattering, given the Earth, the Sun, and the atmosphere medium parameters.

Semi-Empirical Physics Models In many engineering scenarios, the underlying physical process is either not fully understood, or too complex to be efficiently computed. In such cases, semi-empirical physics models are used. We implement two popular semi-empirical models, namely the magic formula for tire dynamics (Pacejka and Bakker, 2026), and the Thevenin equivalent circuit model for battery SoC (State of Charge) estimation (Hannan et al., 2017).

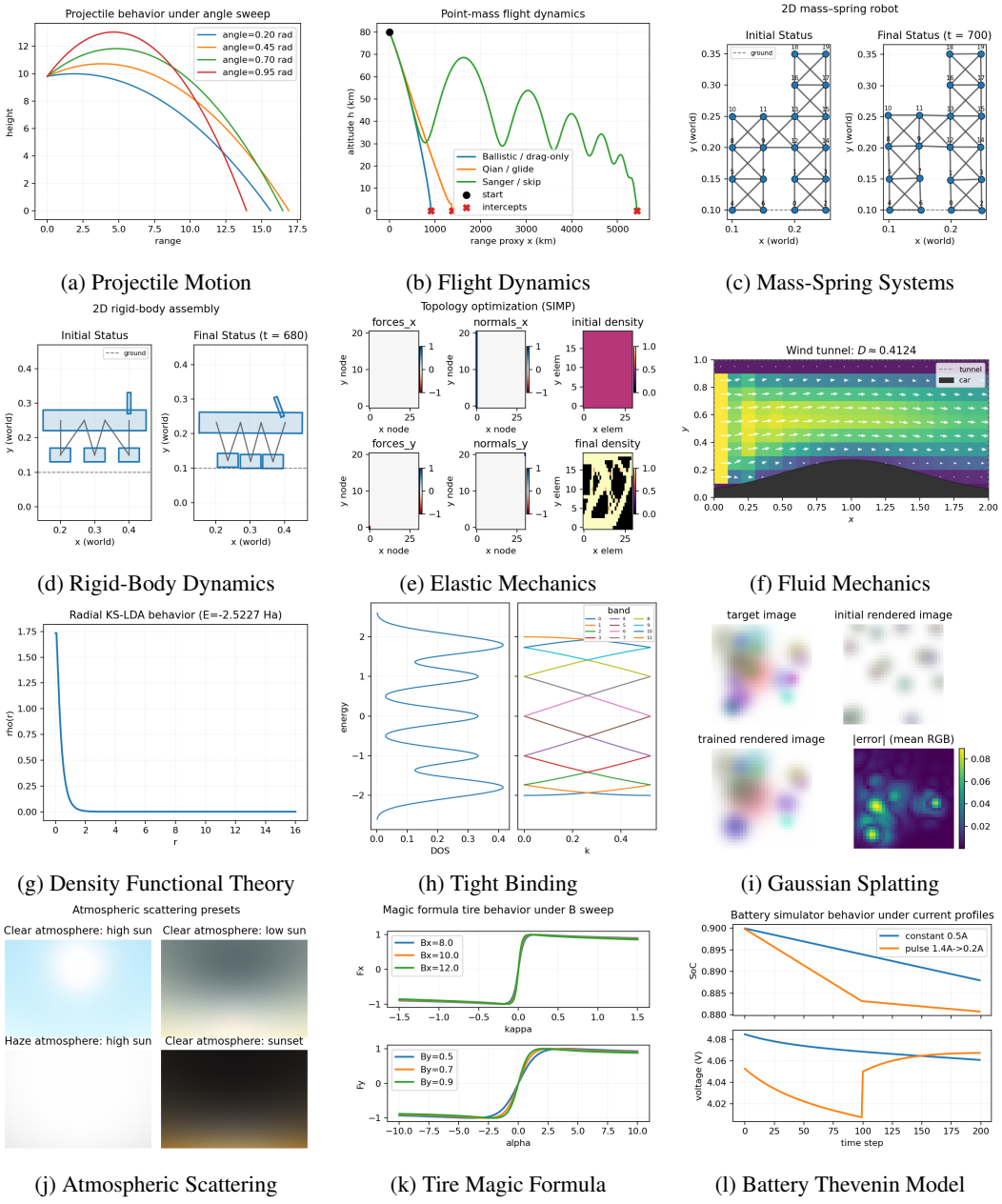
Table 1: List of the implemented differentiable physics models.

Category	Model	Input	Output
Discrete Mechanics	Projectile Motion	Earth's gravity initial state	horizontal range
	Flight Dynamics	vehicle params Earth params initial state control policy	flight trajectory
	Mass-Spring Systems	system params initial state control policy	movement trajectory
	Rigid-Body Dynamics	system params initial state control policy	movement trajectory
Continuous Mechanics	Elastic Mechanics	material density boundary conditions physical params	compliance displacement
	Fluid Mechanics	car shape physical params	drag
Atomistic Simulations	Density Functional Theory	system params	total energy electron density
	Tight Binding	material params	eigenvalue spectrum Bloch bands
Rendering	Gaussian Splatting	scene params render params	scene image
	Atmospheric Scattering	Earth params Sun params atmosphere params render params	sky image
Semi-Empirical	Tire Magic Formula	tire params	tire forces
	Battery Thevenin Model	circuit params OCV params initial state current profile	voltage SoC trajectory

3.2 Inverse Problems

We design inverse problems based on the constructed differentiable physics models, which can be categorized into 3 types, namely parameter identification, inverse design and optimal control. The dimensions of the optimization variables of different problems are designed to be of different orders of magnitude. We intentionally keep the problems small in scale, so that each optimizer can be benchmarked with all inverse problems within a reasonable time. Also, we believe the performance on larger problems can be estimated from the results on small problems, as they share the same numerical characteristics. The inverse problem tasks are summarized in Table 2.

Parameter Identification For parameter identification problems, the goal is to identify the unknown parameters in the physics model so that it best matches observed data. For the XC Functional Fitting problem, we fit the LDA exchange-correlation functional to the reference total energy and electron density. For the Tire Params Fitting problem, we fit the Pacejka BCDE parameters so longitudinal and lateral forces match synthetic curves from a hidden ground-truth. For the Battery Params Fitting problem, we fit the first-order Thevenin circuit parameters together with a cubic open-circuit



(a) Projectile Motion (b) Flight Dynamics (c) Mass-Spring Systems

(d) Rigid-Body Dynamics (e) Elastic Mechanics (f) Fluid Mechanics

(g) Density Functional Theory (h) Tight Binding (i) Gaussian Splatting

(j) Atmospheric Scattering (k) Tire Magic Formula (l) Battery Thevenin Model

Figure 1: Gallery of the implemented differentiable physics simulators.

voltage map so that, under fixed random pulse-current profiles and initial state of charge, the predicted terminal voltage and state-of-charge trajectories match a frozen ground-truth battery. For the Tight Binding Params Fitting problem, we fit onsite and hopping parameters so that the supercell eigenvalue spectrum and the Bloch band energies match a frozen reference chain, with a larger weight on the eigenvalues than on the bands. For the 2D Reconstruction problem, we fit the Gaussian blobs for the rendered image to match a target image. More detailed and analytical descriptions of the above problems are provided in Appendix B.1.

Inverse Design For inverse design problems, the goal is to optimize certain parameters of the system so that a desired performance metric can achieve its best. The Range Maximization problem of Projectile Motion adjusts the launch angle so that the horizontal range is maximized. The Shape Optimization problem tunes the shape parameters of the car silhouette in order to minimize the

Table 2: List of the benchmarked inverse problems.

Category	Problem	Base Simulator	Dimension of Design Variables
Parameter Identification	XC Functional Fitting	DFT	2
	Tire Params Fitting	Tire Magic Formula	8
	Battery Params Fitting	Battery Thevenin Model	9
	Tight Binding Params Fitting	Tight Binding	32
	2D Reconstruction	Gaussian Splatting	224
Inverse Design	Range Maximization	Projectile Motion	1
	Shape Optimization	Fluid Mechanics	4
	Chromaticity Maximization	Atmospheric Scattering	15
	Topology Optimization	Elastic Mechanics	600
Optimal Control	Robot Setpoint	Rigid-Body Dynamics	70
	Range Maximization	Flight Dynamics	400
	Robot Reach	Mass-Spring Systems	1100

drag estimate. The Chromaticity Maximization problem is to discover the best sun direction and atmosphere medium parameters to maximize the RGB difference between the top and bottom of the rendered sky image. The Topology Optimization problem optimizes element densities on a half MBB beam to minimize scaled compliance subject to a soft volume constraint that keeps the mean density near a prescribed volume fraction. More detailed and analytical descriptions of the above problems are provided in Appendix B.2.

Optimal Control The optimal control problems are only for dynamic systems, where an optimal policy can be discovered for the system to achieve optimum of a desired performance. The Range Maximization problem of Flight Dynamics learns an open-loop schedule of lift and drag coefficients over a fixed horizon. The Robot Setpoint problem with Rigid-Body Dynamics learns open-loop spring actuations over a short horizon to drive a designated body toward a planar goal, with terminal velocity and control-regularization terms. The Robot Reach problem with Mass-Spring Systems likewise learns open-loop spring actuations over a longer horizon to reach a planar goal, with analogous velocity and effort or smoothness penalties. More detailed and analytical descriptions of the above problems are provided in Appendix B.3.

3.3 Optimizers

The optimizers we benchmark fall into 3 categories, namely first-order local optimizers, approximate second-order local optimizers, and global optimizers utilizing gradient information for local search. For each category, we implemented 4 optimizers, as listed in Table 3. For first-order local optimizers, they are the most commonly used optimizers in deep learning. In this work, we include PyTorch official implementations of SGD, RMSprop, Adam, and AdamW optimizers. For approximate second-order local optimizers, they have better convergence and accuracy properties than first-order optimizers, hence are suitable for scientific computing problems. Except for the classical L-BFGS method, the other 3 optimizers are modern variants that are specifically designed for deep learning, i.e., K-FAC (Martens and Grosse, 2015), Shampoo (Gupta et al., 2018) and SOAP (Vyas et al., 2025). The above two

Table 3: List of the benchmarked optimizers.

Category	Optimizer
First-order Local	SGD
	RMSprop
	Adam
	AdamW
Approximate Second-order Local	L-BFGS
	K-FAC
	Shampoo
	SOAP
Global	Random Search
	Simulated Annealing
	Particle Swarm
	Differential Evolution

Table 4: The performance of optimizers on the benchmarked parameter identification problems.

Optimizer	DFT ↓	Tire ↓	Battery ↓	Tight Binding ↓	Gaussian Splatting ↓
SGD	$1.9e^{-14} \pm 2.7e^{-14}$	$3.8e^{-4} \pm 1.9e^{-5}$	$2.0e^{-5} \pm 1.4e^{-5}$	$3.2e^{-5} \pm 2.1e^{-6}$	$2.2e^{-2} \pm 4.7e^{-4}$
RMSprop	$\mathbf{0.0 \pm 0.0}$	$2.6e^{-4} \pm 3.7e^{-5}$	$2.5e^{-4} \pm 2.2e^{-4}$	$5.6e^{-5} \pm 1.2e^{-5}$	$1.6e^{-2} \pm 1.6e^{-4}$
Adam	$1.9e^{-14} \pm 2.7e^{-14}$	$2.6e^{-4} \pm 1.7e^{-5}$	$5.1e^{-5} \pm 3.3e^{-5}$	$7.2e^{-7} \pm 2.8e^{-7}$	$1.4e^{-2} \pm 8.8e^{-5}$
AdamW	$3.5e^{-12} \pm 3.0e^{-12}$	$3.4e^{-4} \pm 1.5e^{-5}$	$3.3e^{-4} \pm 3.6e^{-4}$	$8.5e^{-7} \pm 4.9e^{-7}$	$1.4e^{-2} \pm 1.6e^{-4}$
LBFGS	$8.8e^{-11} \pm 6.2e^{-11}$	$\mathbf{2.5e^{-7} \pm 4.0e^{-8}}$	$\mathbf{3.3e^{-6} \pm 5.2e^{-7}}$	$\mathbf{1.1e^{-9} \pm 5.2e^{-10}}$	$\mathbf{1.0e^{-2} \pm 1.9e^{-4}}$
KFAC	$5.1e^{-10} \pm 1.3e^{-10}$	$2.4e^{-4} \pm 1.9e^{-5}$	$4.9e^{-5} \pm 5.8e^{-5}$	$3.5e^{-7} \pm 6.0e^{-8}$	$2.2e^{-2} \pm 1.1e^{-3}$
Shampoo	$\mathbf{0.0 \pm 0.0}$	$5.7e^{-4} \pm 1.0e^{-4}$	$4.3e^{-4} \pm 5.4e^{-4}$	$6.9e^{-4} \pm 2.2e^{-4}$	$1.4e^{-2} \pm 1.0e^{-4}$
SOAP	$3.2e^{-9} \pm 1.2e^{-9}$	$2.5e^{-4} \pm 1.5e^{-5}$	$5.6e^{-5} \pm 2.4e^{-5}$	$4.1e^{-7} \pm 8.9e^{-8}$	$1.5e^{-2} \pm 5.2e^{-4}$
RS	$1.9e^{-14} \pm 2.7e^{-14}$	$4.2e^{-4} \pm 3.2e^{-5}$	$3.8e^{-5} \pm 1.1e^{-6}$	$1.3e^{-4} \pm 1.6e^{-5}$	$1.9e^{-2} \pm 3.4e^{-4}$
SA	$9.3e^{-11} \pm 2.2e^{-11}$	$1.1e^{-3} \pm 3.7e^{-4}$	$4.5e^{-5} \pm 2.5e^{-6}$	$5.2e^{-3} \pm 4.9e^{-4}$	$2.2e^{-2} \pm 5.0e^{-4}$
PS	$\mathbf{0.0 \pm 0.0}$	$8.1e^{-5} \pm 2.0e^{-5}$	$8.5e^{-6} \pm 1.5e^{-6}$	$2.0e^{-5} \pm 7.1e^{-6}$	$1.9e^{-2} \pm 4.4e^{-4}$
DE	$\mathbf{0.0 \pm 0.0}$	$1.3e^{-4} \pm 4.2e^{-5}$	$1.7e^{-5} \pm 5.9e^{-6}$	$4.4e^{-3} \pm 7.7e^{-4}$	$2.6e^{-2} \pm 1.4e^{-3}$

Table 5: The performance of optimizers on the benchmarked inverse design problems.

Optimizer	Projectile ↑	Fluid ↓	Atmospheric ↑	Elastic ↓
SGD	$\mathbf{1.7e^{1} \pm 0.0}$	$6.2e^{-2} \pm 4.6e^{-7}$	$7.5e^{-1} \pm 2.0e^{-3}$	$1.5e^{0} \pm 4.8e^{-3}$
RMSprop	$\mathbf{1.7e^{1} \pm 0.0}$	$4.2e^{-2} \pm 1.7e^{-7}$	$7.5e^{-1} \pm 9.6e^{-5}$	$1.6e^{0} \pm 5.5e^{-3}$
Adam	$\mathbf{1.7e^{1} \pm 0.0}$	$4.0e^{-2} \pm 1.3e^{-7}$	$7.5e^{-1} \pm 6.2e^{-4}$	$1.5e^{0} \pm 5.2e^{-3}$
AdamW	$1.7e^{1} \pm 9.0e^{-7}$	$4.2e^{-2} \pm 2.3e^{-7}$	$7.4e^{-1} \pm 1.9e^{-3}$	$1.5e^{0} \pm 6.0e^{-3}$
LBFGS	$\mathbf{1.7e^{1} \pm 0.0}$	$4.0e^{-2} \pm 2.1e^{-8}$	$\mathbf{7.5e^{-1} \pm 4.9e^{-8}}$	$\mathbf{1.5e^{0} \pm 4.5e^{-3}}$
KFAC	$1.7e^{1} \pm 9.0e^{-7}$	$4.0e^{-2} \pm 9.8e^{-9}$	$7.5e^{-1} \pm 2.1e^{-5}$	$1.6e^{0} \pm 1.2e^{-2}$
Shampoo	$\mathbf{1.7e^{1} \pm 0.0}$	$\mathbf{4.0e^{-2} \pm 0.0}$	$7.5e^{-1} \pm 6.7e^{-4}$	$1.6e^{0} \pm 8.7e^{-3}$
SOAP	$1.7e^{1} \pm 9.0e^{-7}$	$4.0e^{-2} \pm 7.3e^{-6}$	$7.5e^{-1} \pm 4.8e^{-5}$	$1.9e^{0} \pm 3.5e^{-2}$
RS	$1.7e^{1} \pm 1.6e^{-6}$	$\mathbf{4.0e^{-2} \pm 0.0}$	$7.2e^{-1} \pm 4.0e^{-5}$	$1.5e^{0} \pm 1.6e^{-2}$
SA	$\mathbf{1.7e^{1} \pm 0.0}$	$\mathbf{4.0e^{-2} \pm 0.0}$	$7.2e^{-1} \pm 2.0e^{-2}$	$2.6e^{0} \pm 5.7e^{-2}$
PS	$\mathbf{1.7e^{1} \pm 0.0}$	$\mathbf{4.0e^{-2} \pm 0.0}$	$7.5e^{-1} \pm 5.6e^{-8}$	$1.5e^{0} \pm 6.6e^{-3}$
DE	$\mathbf{1.7e^{1} \pm 0.0}$	$4.0e^{-2} \pm 1.8e^{-9}$	$7.5e^{-1} \pm 1.3e^{-4}$	$2.5e^{0} \pm 2.0e^{-2}$

categories of optimizers are all local optimizers, which means they are only capable of converging to local minima. However for problems with non-convex loss landscape and optimization variables with physical meanings, it is often desirable to find the global minimum. Hence we also include 4 well-recognized global optimizers, namely Random Search (Bergstra and Bengio, 2012), Simulated Annealing (Kirkpatrick et al., 1983), Particle Swarm (Kennedy and Eberhart, 1995), and Differential Evolution (Das and Suganthan, 2010), which utilize gradient information for local search, but also have mechanisms to escape from local minima and explore the global landscape.

4 Evaluation and Analysis

In this section we benchmark the optimizers with the designed inverse problem tasks on the implemented differentiable physics models. For each task and optimizer, we tune optimizer hyperparameters on a predefined shared grid, select the best configuration by minimizing the mean best loss over 3 random initializations, and report mean±std of the best objective across those runs. Non-converged runs are excluded before hyper-parameter selection and aggregation. All experiments are performed on a server with an NVIDIA A100 GPU (80 GB). We record the optimization process, time and memory consumption during experiments. The major results are summarized in Table 4–6. In the table, RS is short for Random Search, SA is short for Simulated Annealing, PS is short for Particle Swarm, and DE is short for Differential Evolution. More details regarding experimental setups and results are provided in Appendix C.

Table 6: The performance of optimizers on the benchmarked optimal control problems.

Optimizer	Flight Dynamics ↑	Rigid Body ↓	Mass Spring ↓
SGD	$1.3\text{e}^{0} \pm 1.2\text{e}^{-6}$	$1.9\text{e}^{-1} \pm 1.7\text{e}^{-5}$	$1.9\text{e}^{-1} \pm 1.1\text{e}^{-2}$
RMSprop	$1.3\text{e}^{0} \pm 6.7\text{e}^{-7}$	$1.9\text{e}^{-1} \pm 1.3\text{e}^{-5}$	$5.9\text{e}^{-2} \pm 9.6\text{e}^{-3}$
Adam	$1.3\text{e}^{0} \pm 9.7\text{e}^{-8}$	$1.9\text{e}^{-1} \pm 2.7\text{e}^{-5}$	$1.3\text{e}^{-1} \pm 1.3\text{e}^{-2}$
AdamW	$1.3\text{e}^{0} \pm 3.5\text{e}^{-7}$	$1.9\text{e}^{-1} \pm 1.5\text{e}^{-5}$	$1.4\text{e}^{-1} \pm 1.9\text{e}^{-2}$
LBFGS	$1.3\text{e}^{0} \pm 5.0\text{e}^{-6}$	$1.9\text{e}^{-1} \pm 1.6\text{e}^{-3}$	$2.0\text{e}^{-1} \pm 3.0\text{e}^{-2}$
KFAC	$1.3\text{e}^{0} \pm 5.6\text{e}^{-8}$	$1.9\text{e}^{-1} \pm 7.8\text{e}^{-6}$	$1.9\text{e}^{-1} \pm 1.0\text{e}^{-2}$
Shampoo	$\mathbf{1.3\text{e}^{0} \pm 0.0}$	$\mathbf{1.9\text{e}^{-1} \pm 2.5\text{e}^{-6}}$	$\mathbf{5.8\text{e}^{-2} \pm 1.0\text{e}^{-2}}$
SOAP	$1.3\text{e}^{0} \pm 2.6\text{e}^{-3}$	$1.9\text{e}^{-1} \pm 1.1\text{e}^{-5}$	$1.1\text{e}^{-1} \pm 8.4\text{e}^{-3}$
RS	$1.3\text{e}^{0} \pm 2.2\text{e}^{-4}$	$1.9\text{e}^{-1} \pm 4.1\text{e}^{-6}$	$9.5\text{e}^{-2} \pm 1.6\text{e}^{-2}$
SA	$1.3\text{e}^{0} \pm 1.8\text{e}^{-3}$	$1.9\text{e}^{-1} \pm 2.0\text{e}^{-5}$	$1.1\text{e}^{-1} \pm 1.9\text{e}^{-2}$
PS	$1.3\text{e}^{0} \pm 1.1\text{e}^{-3}$	$1.9\text{e}^{-1} \pm 3.6\text{e}^{-5}$	$1.0\text{e}^{-1} \pm 3.6\text{e}^{-2}$
DE	$1.3\text{e}^{0} \pm 9.2\text{e}^{-4}$	$1.9\text{e}^{-1} \pm 1.4\text{e}^{-4}$	$1.1\text{e}^{-1} \pm 5.7\text{e}^{-3}$

When curvature helps, and when it hurts. Across parameter identification (Table 4), approximate second-order methods are not uniformly better, but they decisively shine on structured spectral matching: L-BFGS reaches a mean best loss on the order of 10^{-9} on tight-binding fitting, with K-FAC and SOAP close behind at $\sim 10^{-7}$, whereas untuned first-order baselines such as SGD plateau several orders of magnitude higher. This matches the intuition that least-squares-type objectives against smooth matrix spectra are locally analytic and benefit from exploiting gradient geometry. Conversely, on the radial LDA exchange–correlation fit, nearly all optimizer families reach near-zero best losses (many within numerical tolerance of 0) once hyperparameters are selected per method, indicating a well-conditioned two-parameter fit when the forward SCF remains stable. Tight-binding and DFT therefore bracket a key lesson for differentiable physics benchmarks: optimizer rankings are most informative on tasks where the forward model yields stable objectives but the landscape still separates update rules.

Global search, initialization sensitivity, and dimensionality. The expectation that global optimizers can be more robust under random initialization is supported most cleanly on low-dimensional inverse design. On projectile range maximization (Table 5), all twelve optimizers reach essentially the same optimal best objective (mean ≈ 16.97), indicating repeated landing on the global maximum of this one-dimensional landscape. The classical trade-off remains: when the basin is generous, local and global methods alike succeed; when it is not, population-based methods pay extra forward passes but buy insurance against bad seeds.

The topology-optimization column is the counterpoint: with hundreds of design variables, most first-order methods and several global methods attain stable mean best losses around 1.5, while simulated annealing and differential evolution plateau higher near 2.5–2.6. Without a convergence checker, a minority of K-FAC and differential-evolution runs collapse to large negative losses (about 3% of all SIMP runs, concentrated in these two optimizers); naïve minimization of mean best loss would then prefer those failed trajectories. After discarding non-converged runs and re-selecting hyperparameters on converged repeats only, K-FAC lands with the stable cluster (~ 1.6), whereas differential evolution remains among the weaker but finite performers. Each compliance evaluation requires solving a large sparse equilibrium system inside the autodiff graph, so high-variance proposals can still probe physically meaningless density fields—but such failures should be filtered rather than rewarded. This supports the following conclusion that, global optimizers improve robustness primarily when either (i) the search space is low-dimensional, or (ii) stochastic gradient noise dominates and local curvature is uninformative—not automatically when the forward model is "physics-based."

Rendering and semi-empirical fits. Gaussian splatting reconstruction (224 parameters) shows a clear split: L-BFGS achieves the strongest mean best loss around 10^{-2}, adaptive first-order methods land nearby, and global methods remain slightly worse around 2×10^{-2} with small run-to-run variance. Semi-empirical tire and battery fits are comparatively forgiving: most methods land in a

band of 10^{-6}–10^{-3} mean best losses, with L-BFGS particularly strong on both tasks and particle swarm competitive on the tire task. These tasks behave closer to "standard" differentiable programming in machine learning, where tuned first-order and second-order methods are strong baselines and globals serve as sanity checks rather than sole workhorses.

Optimal control: long unrolls and noisy credit assignment. The flight-dynamics and mass–spring columns (Table 6) exemplify the point about backpropagating through long rollouts. Flight dynamics now yields tightly clustered mean best scores around 1.3 with very small standard deviations across all twelve optimizers, indicating a comparatively well-conditioned open-loop objective at the reported scale, with only modest separation among update rules. Mass–spring control remains more discriminative (mean best losses ~ 0.06–0.20), with Shampoo and RMSprop among the strongest configurations, suggesting that even stable contact-rich rollouts retain multi-modal structure in the policy space. Rigid-body setpoint control in Table 6 attains finite mean best losses for all twelve optimizers, clustered around ~ 0.19 with very small spread; this is qualitatively similar to the tight flight cluster and tighter than mass–spring, indicating a relatively stable setpoint objective here despite contacts and joints, while still leaving limited room for optimizer choice.

Compute–objective trade-offs. Appendix C.2 makes the runtime story quantitative. On DFT XC fitting, a single global-optimizer step costs roughly 5–1.9×10^1 s of wall time versus ~ 1 s for first- and second-order local methods—an order-of-magnitude gap before counting how many steps each family needs. The ratio is milder on inexpensive forward models (tire, Gaussian splatting) but widens again on elastic topology optimization and flight dynamics, where one global step can exceed ten forward–backward passes of a local optimizer. We therefore come to the following conclusion that, for offline calibration, design, or scientific inversion, paying an order of magnitude in wall time can be rational if it reduces residual error or avoids manual initialization engineering; for real-time control or any setting where the optimizer sits inside an inner loop at kHz rates, first-order methods with small per-step cost remain the pragmatic default, and the benchmark encourages reporting both objective and step-time tables whenever a new simulator is added.

Takeaway for future optimizer research. Differentiable physics does not inherit the "default optimizers" culture of deep networks: the same Adam hyperparameters that suffice for splatting can be less decisive for a Kohn–Sham outer loop, while L-BFGS may dominate on a 32-parameter tight-binding ring yet fail to offset the cost of global search on a 600-variable topology field. We hope the zoo encourages methods that adapt to simulator structure—mixed-precision stable eigensolvers for quantum tasks, trust-region filters for contact rollouts, and hybrid global-local schedules whose search radius shrinks as physics residuals fall—rather than treating optimizer choice as an afterthought bolted onto an otherwise frozen forward model.

5 Conclusion

In this work, we constructed to the best of our knowledge the first benchmark for optimizers to solve inverse problems with differentiable physics simulators. More specifically, we implemented 12 differentiable physics simulators across various physics domains, and designed corresponding inverse problems spanning parameter identification, inverse design, and optimal control. For optimizers, we considered first-order, approximate second-order, and global optimizers, all utilizing the gradient information provided by the differentiable physics models and automatic differentiation tools. We benchmarked the performance of optimizers on these inverse problems, and analyzed the results to provide insights on how to choose and design optimizers for differentiable programming.

Limitations and Broader impacts. For inverse problem task types, besides the 3 types we benchmarked in this work, there are also other types of tasks that can be solved by differentiable physics models, such as data assimilation and state estimation. Also, constrained optimization has not been considered in the current work, which is also ubiquitous in real-life applications. We will develop more physics simulators, task types and optimizers continuously. Broader impacts of this work include the potential to further promote the applications of differentiable programming in various scientific and engineering domains.

AI USE STATEMENT

In this work, we used generative AI tools (LLM-based coding and writing assistants) for implementing parts of the benchmark codebase and running experiments. We also used generative AI tools for drafting or editing portions of the manuscript. We have not used generative AI tools to search for related works, generate synthetic datasets, or prove mathematical claims. We have reviewed all AI-assisted work. The generated code was either manually checked or tested with numerical experiments and visualization. The generated text was all reviewed and further edited by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work is about differentiable physics simulations and solving inverse problems, hence does not involve human subjects, personally identifiable information, potentially harmful insights, discrimination/bias/fairness concerns, privacy and security issues, or legal compliance. No potential conflicts of interest and sponsorship are related.

REPRODUCIBILITY STATEMENT

We have made the following efforts to support reproducibility. The full codebase—including the differentiable simulators, inverse-problem tasks, optimizer implementations, and scripts to run the hyper-parameter grid search and experiments—is submitted with this paper as Supplementary Material (anonymous during review; public release upon acceptance). Appendix A documents the mathematical formulations of the simulators; Appendix B specifies the inverse problems and their loss functions; Appendix C specifies the experimental protocol (grid size, repeats, epochs, initialization noise, hardware, etc.) together with wall-time and peak GPU memory statistics.

REFERENCES

Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks*, 11(1):15–37, 1998.

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–43, 2018.

Lenaic Chizat and Francis Bach. Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. In *Conference on learning theory*, pages 1305–1338. PMLR, 2020.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

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.

Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *IEEE transactions on pattern analysis and machine intelligence*, 45(8):10173–10196, 2023.

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.

Philipp Holl and Nils Thuerey. Phiflow: Differentiable simulations for pytorch, tensorflow and jax. In *International Conference on Machine Learning*. PMLR, 2024.

J. Zico Kolter, David Duvenaud, and Matthew Johnson. Deep implicit layers: Neural odes, deep equilibrium models, and beyond. NeurIPS 2020 Tutorial, 2020. URL `https://implicit-layers-tutorial.org/`.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Sebastian Mitusch, Simon Funke, and Jørgen Dokken. dolfin-adjoint 2018.1: automated adjoints for fenics and firedrake. *Journal of Open Source Software*, 4(38):1292, 2019.

Andrin Rehmann, Heiko Zimmermann, and Dion Häfner. Mosaic: A benchmark suite for differentiable physics solvers. *arXiv preprint arXiv:2606.27895*, 2026.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pages 136–145. PMLR, 2017.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

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.

Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2019.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

Stephan Hoyer, Jascha Sohl-Dickstein, and Sam Greydanus. Neural reparameterization improves structural optimization. In *NeurIPS 2019 Workshop on Solving Inverse Problems with Deep Networks*, 2019.

Ardavan F Oskooi, David Roundy, Mihai Ibanescu, Peter Bermel, John D Joannopoulos, and Steven G Johnson. Meep: A flexible free-software package for electromagnetic simulations by the fdtd method. *Computer Physics Communications*, 181(3):687–702, 2010.

Shane Colburn and Arka Majumdar. Inverse design and flexible parameterization of meta-optics using algorithmic differentiation. *Communications Physics*, 4(1):65, 2021.

Xutong Jin, Chenxi Xu, Ruohan Gao, Jiajun Wu, Guoping Wang, and Sheng Li. Diffsound: Differentiable modal sound rendering and inverse rendering for diverse inference tasks. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–12, 2024.

Li Li, Stephan Hoyer, Ryan Pederson, Ruoxi Sun, Ekin D Cubuk, Patrick Riley, and Kieron Burke. Kohn-sham equations as regularizer: Building prior knowledge into machine-learned physics. *Physical review letters*, 126(3):036401, 2021.

M. F. Kasim and S. M. Vinko. Learning the exchange-correlation functional from nature with fully differentiable density functional theory. *Phys. Rev. Lett.*, 127:126403, Sep 2021. doi: 10.1103/PhysRevLett.127.126403.

Stefan Doerr, Maciej Majewski, Adrià Pérez, Andreas Kramer, Cecilia Clementi, Frank Noe, Toni Giorgino, and Gianni De Fabritiis. Torchmd: A deep learning framework for molecular simulations. *Journal of chemical theory and computation*, 17(4):2355–2363, 2021.

Teresa Tamayo-Mendoza, Christoph Kreisbeck, Roland Lindh, and Alán Aspuru-Guzik. Automatic differentiation in quantum chemistry with applications to fully variational hartree–fock. *ACS central science*, 4(5):559–566, 2018.

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.

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. *Communications of the ACM*, 65(1):99–106, 2021.

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

Samuel Schoenholz and Ekin Dogus Cubuk. Jax md: a framework for differentiable physics. *Advances in Neural Information Processing Systems*, 33:11428–11441, 2020.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Tianju Xue, Shuheng Liao, Zhengtao Gan, Chanwook Park, Xiaoyu Xie, Wing Kam Liu, and Jian Cao. Jax-fem: A differentiable gpu-accelerated 3d finite element solver for automatic inverse design and mechanistic data science. *Computer Physics Communications*, 291:108802, 2023.

Jean-Eric Campagne, François Lanusse, Joe Zuntz, Alexandre Boucaud, Santiago Casas, Minas Karamanis, David Kirkby, Denise Lanzieri, Yin Li, and Austin Peel. Jax-cosmo: An end-to-end differentiable and gpu accelerated cosmology library. *arXiv preprint arXiv:2302.05163*, 2023.

Honghui Du and QiZhi He. Jax-mpm: A learning-augmented differentiable meshfree framework for gpu-accelerated lagrangian simulation and geophysical inverse modeling. *arXiv preprint arXiv:2507.04192*, 2025.

Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckas, Elliot Saba, Viral B Shah, and Will Tebbutt. A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587*, 2019.

Ruo-Yu Sun. Optimization for deep learning: An overview. *Journal of the Operations Research Society of China*, 8(2):249–294, 2020.

Frank Schneider, Lukas Balles, and Philipp Hennig. Deepobs: A deep learning optimizer benchmark suite. *arXiv preprint arXiv:1903.05499*, 2019.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL `https://kellerjordan.github.io/posts/muon/`.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Raghu Bollapragada, Jorge Nocedal, Dheevatsa Mudigere, Hao-Jun Shi, and Ping Tak Peter Tang. A progressive batching l-bfgs method for machine learning. In *International Conference on Machine Learning*, pages 620–629. PMLR, 2018.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR, 2018.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade. Soap: Improving and stabilizing shampoo using adam for language modeling. In *The Thirteenth International Conference on Learning Representations*, 2025.

Peter W Christensen and Anders Klarbring. *An introduction to structural optimization*, volume 153. Springer Science & Business Media, 2008.

Philippe Angot, Charles-Henri Bruneau, and Pierre Fabrie. A penalization method to take into account obstacles in incompressible viscous flows. *Numerische Mathematik*, 81(4):497–520, 1999.

F Matthias Bickelhaupt and Evert Jan Baerends. Kohn-sham density functional theory: predicting and understanding chemistry. *Reviews in computational chemistry*, pages 1–86, 2000.

CM Goringe, DR Bowler, and E Hernández. Tight-binding modelling of materials. *Reports on Progress in Physics*, 60(12):1447–1512, 1997.

John William Strutt Baron Rayleigh. *On the scattering of light by small particles*. 1871.

Hans B Pacejka and Egbert Bakker. The magic formula tyre model. In *Tyre models for vehicle dynamics analysis*, pages 1–18. CRC Press, 2026.

Mohammad A Hannan, MS Hossain Lipu, Aini Hussain, and Azah Mohamed. A review of lithium-ion battery state of charge estimation and management system in electric vehicle applications: Challenges and recommendations. *Renewable and Sustainable Energy Reviews*, 78:834–854, 2017.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2010.

A Details of Differentiable Physics Model Zoo

A.1 Discrete Mechanics

Projectile Motion. We model planar motion of a point mass under constant gravitational acceleration $g > 0$, with initial height $h_0 > 0$, launch speed $v_0 > 0$, and launch angle θ. The initial velocity components are $v_{x,0} = v_0 \cos\theta$ and $v_{y,0} = v_0 \sin\theta$. The trajectory is the parabola:

$$x(t) = v_{x,0}t, \qquad y(t) = h_0 + v_{y,0}t - \tfrac{1}{2}gt^2. \tag{1}$$

The simulator is implemented in closed form: the time of flight to ground impact is obtained by solving $y(t) = 0$ for the positive root,

$$t_{\text{imp}} = \frac{v_{y,0} + \sqrt{v_{y,0}^2 + 2gh_0}}{g}, \tag{2}$$

and the horizontal range used as the scalar output is $s_x = v_{x,0}t_{\text{imp}}$.

Flight Dynamics. The vehicle is modeled as a point mass m with reference area S. The system state is set as $\mathbf{s} = (h, x, v, \gamma)^\top$: altitude h (m), a ground-range coordinate x (m), speed v (m/s), and flight-path angle γ (rad, from horizontal). At each step, a control policy returns aerodynamic coefficients (C_L, C_D) as functions of $\mathbf{s}$. Let R_e denote the Earth radius and μ denote the gravitational parameter. The radial distance $r = R_e + h$ gives $g(r) = \mu/r^2$. The air density $\rho(h) = \rho_0 \exp(-h/H)$ for $h \le 120\,\text{km}$ and $\rho = 0$ above. The dynamic pressure is $q = \frac{1}{2}\rho(h)v^2$, with lift $L = qSC_L$ and drag $D = qSC_D$. The system state is updated by:

$$\dot{h} = v \sin\gamma, \tag{3}$$

$$\dot{x} = v \cos\gamma \, \frac{R_e}{r}, \tag{4}$$

$$\dot{v} = -\frac{D}{m} - g(r)\sin\gamma, \tag{5}$$

$$\dot{\gamma} = \frac{L}{mv} - \left(\frac{g(r)}{v} - \frac{v}{r}\right)\cos\gamma. \tag{6}$$

One explicit Euler update with step Δt advances $\mathbf{s} \leftarrow \mathbf{s} + \Delta t\,(\dot{h}, \dot{x}, \dot{v}, \dot{\gamma})^\top$. Rollouts start from a stored initial state and end until $h \le 0$. In the benchmarked open-loop task, (C_L, C_D) are produced from per-step logits as above; positions remain in SI meters inside the integrator, while the loss and trajectory plots divide h and x by $U = 10^6\,\mathrm{m}$ so that the range term $-x_T/U$ is expressed in units of $1000\,\mathrm{km}$, together with the effort/smoothness regularizers scaled by $1/U$.

Mass-Spring Systems. The mathematical formulation and experimental setups are inspired by Hu et al. (2020, Sec. E.5). We consider a two-dimensional particle system with N point masses. Let $\mathbf{x}_i(t),\ \mathbf{v}_i(t) \in \mathbb{R}^2$ denote the position and velocity of mass $i \in \{1, \dots, N\}$. Springs are undirected edges $e = (a, b)$ between anchors $a, b \in \{1, \dots, N\}$. Each spring has a nominal rest length $L_e > 0$, a stiffness $k_e \ge 0$, and an actuation gain $\alpha_e \ge 0$. A scalar control $u_e^n \in [-1, 1]$ is applied at discrete time index n; passive springs correspond to $\alpha_e = 0$, for which u_e^n has no effect. The commanded natural length at step n is $L_e^{\mathrm{tar}}(u_e^n) \;=\; L_e\,\big(1 + \alpha_e\,u_e^n\big)$. Let $\mathbf{d}_e^n \;=\; \mathbf{x}_a^n - \mathbf{x}_b^n$ and $\ell_e^n = \|\mathbf{d}_e^n\|$. For numerical stability we use a regularized length $\tilde{\ell}_e^n = \max(\ell_e^n, \varepsilon)$ with fixed small $\varepsilon > 0$. The spring exerts a velocity impulse aligned with $\mathbf{d}_e^n$:

$$\mathbf{\Delta v}_e^n \;=\; \Delta t\,k_e\,\frac{\tilde{\ell}_e^n - L_e^{\mathrm{tar}}(u_e^n)}{\tilde{\ell}_e^n}\,\mathbf{d}_e^n, \tag{7}$$

which is accumulated on the endpoints as $\mathbf{v}_a \leftarrow \mathbf{v}_a - \mathbf{\Delta v}_e^n$ and $\mathbf{v}_b \leftarrow \mathbf{v}_b + \mathbf{\Delta v}_e^n$.

Let $\zeta > 0$ be a damping coefficient and $g < 0$ a constant gravitational acceleration along the vertical axis (so the gravitational increment to velocity is $(0,\, g\,\Delta t)$). After summing spring impulses, velocities are updated by exponential damping and gravity:

$$\widetilde{\mathbf{v}}_i^{n+1} \;=\; e^{-\zeta\,\Delta t}\,\mathbf{v}_i^n \;+\; \big(0,\, g\,\Delta t\big) \;+\; \sum_{e \ni i} \delta\mathbf{v}_{e,i}^n, \tag{8}$$

where $\delta\mathbf{v}_{e,i}^n$ is the contribution of spring e to mass i from the impulse rule above.

Let y denote the vertical component and let the ground be the horizontal line $y = h$. If the pre-update configuration is below the ground, $y_i^n < h$, and the post-impulse vertical velocity is still directed into the ground, $\widetilde{v}_{i,y}^{n+1} < 0$, we apply an inelastic stop projection:

$$\mathbf{v}_i^{n+1} \;=\; \begin{cases} \mathbf{0}, & y_i^n < h \;\text{ and }\; \widetilde{v}_{i,y}^{n+1} < 0, \\ \widetilde{\mathbf{v}}_i^{n+1}, & \text{otherwise.} \end{cases} \tag{9}$$

Positions are then advanced in semi-implicit Euler: $\mathbf{x}_i^{n+1} \;=\; \mathbf{x}_i^n + \Delta t\,\mathbf{v}_i^{n+1}$.

Rigid-Body Dynamics. The mathematical formulation and experimental setups are inspired by Hu et al. (2020, Sec. E.7). The simulator tracks N planar rigid bodies. For each body $i \in \{1, \dots, N\}$ at discrete time n, let $\mathbf{x}_i^n \in \mathbb{R}^2$ be the center of mass, $\mathbf{v}_i^n \in \mathbb{R}^2$ the translational velocity, θ_i^n the orientation, and ω_i^n the scalar angular velocity (about the axis normal to the plane). Each body is a centered axis-aligned rectangle in its own frame with half-extents $(h_{x,i}, h_{y,i})$. With uniform areal density $\rho = 1$, the reference implementation uses

$$m_i = 4\,h_{x,i}h_{y,i}, \qquad I_i = \tfrac{4}{3}\,h_{x,i}h_{y,i}\,\big(h_{x,i}^2 + h_{y,i}^2\big), \tag{10}$$

with inverse parameters m_i^{-1} and I_i^{-1} used in impulse updates. Let $R(\theta)$ denote the plane rotation by θ. A body-fixed attachment offset $\mathbf{r}_i^{\mathrm{b}}$ maps to world position $\mathbf{p}_i^n = \mathbf{x}_i^n + R(\theta_i^n)\mathbf{r}_i^{\mathrm{b}}$ and inherits linear velocity

$$\mathbf{v}_{\mathrm{att},i}^n = \mathbf{v}_i^n + \omega_i^n\,\mathbf{J}\,R(\theta_i^n)\mathbf{r}_i^{\mathrm{b}}, \qquad \mathbf{J} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \tag{11}$$

Between bodies a and b, two body-fixed offsets define world anchors $\mathbf{p}_a^n$ and $\mathbf{p}_b^n$ as above. Define $\mathbf{d}^n = \mathbf{p}_a^n - \mathbf{p}_b^n$ and the regularized length $\tilde{\ell}^n = \max\big(\|\mathbf{d}^n\|, \varepsilon\big)$ with fixed $\varepsilon = 10^{-4}$. For a scalar control $u^n \in [-1, 1]$ (clamped), a passive actuation gain $\alpha = 0$ enforces an inactive control channel. For a spring with rest length $L > 0$ and stiffness k, the commanded length is

$$L_{\mathrm{tar}}^n(u^n) = L\,\big(1 + \alpha\,u^n\big). \tag{12}$$

For a joint, one sets $L^n_{\text{tar}} = 0$ so the anchors coincide in the unconstrained limit. The primary Hooke-style velocity impulse aligned with $\mathbf{d}^n$ is

$$\mathbf{J}^n_{\text{spring}} = \Delta t\, k\, \frac{\tilde{\ell}^n - L^n_{\text{tar}}}{\tilde{\ell}^n}\, \mathbf{d}^n. \tag{13}$$

For joints, an additional projection impulse damps the relative anchor velocity $\mathbf{v}^n_{\text{rel}} = \mathbf{v}^n_{\text{att},a} - \mathbf{v}^n_{\text{att},b}$: with $\widehat{\mathbf{n}}^n_{\text{rel}} = \mathbf{v}^n_{\text{rel}} / \max\big(\|\mathbf{v}^n_{\text{rel}}\|, \eta\big)$ and a small $\eta > 0$, and with the usual effective inertia scalar M^n_{eff} assembled from m_a^{-1}, m_b^{-1}, I_a^{-1}, I_b^{-1} and squared projected lever arms $\big(\widehat{\mathbf{n}}^n_{\text{rel}} \times \mathbf{r}^n_a\big)^2$, $\big(\widehat{\mathbf{n}}^n_{\text{rel}} \times \mathbf{r}^n_b\big)^2$ (two-dimensional scalar cross products), the joint contribution is

$$\mathbf{J}^n_{\text{joint}} = \frac{\|\mathbf{v}^n_{\text{rel}}\|}{M^n_{\text{eff}}}\, \widehat{\mathbf{n}}^n_{\text{rel}}, \qquad \mathbf{J}^n = \mathbf{J}^n_{\text{spring}} + \mathbf{J}^n_{\text{joint}}. \tag{14}$$

The link impulse is applied as equal-and-opposite linear impulses at the anchors: body a receives $-\mathbf{J}^n$ and body b receives $+\mathbf{J}^n$, each decomposed into center-of-mass increments

$$\Delta\mathbf{v}^n_a = -\frac{\mathbf{J}^n}{m_a}, \qquad \Delta\omega^n_a = \frac{(\mathbf{p}^n_a - \mathbf{x}^n_a) \times \mathbf{J}^n}{I_a}, \tag{15}$$

and the opposite signs for body b, with $\times$ denoting the scalar z-component of the planar cross product.

For each body, four corners are generated by independent sign patterns on $(\pm h_{x,i}, \pm h_{y,i})$ in body frame. For each corner, let $\boldsymbol{\rho}^n_i$ be the world vector from the center of mass to the corner (after rotation), with corner position $\mathbf{c}^n_i = \mathbf{x}^n_i + \boldsymbol{\rho}^n_i$ and corner velocity $\mathbf{w}^n_i = \mathbf{v}^n_i + \omega^n_i\, \mathbf{J}\boldsymbol{\rho}^n_i$. For the ground test, the velocity is augmented by gravity over one step, $\mathbf{w}^{n,*}_i = \mathbf{w}^n_i + (0, g\Delta t)^\top$, and the predictor position is $\widehat{\mathbf{c}}^n_i = \mathbf{c}^n_i + \Delta t\, \mathbf{w}^{n,*}_i$. With horizontal ground $y = h$ and outward normal $\mathbf{n} = (0,1)^\top$, contact activation uses $\mathbf{n}^\top \mathbf{w}^{n,*}_i < 0$ and $\widehat{c}^n_{i,y} < h$. The normal impulse magnitude $J^n_{n,i} \geq 0$ removes incoming normal motion in the inelastic limit with restitution coefficient e (typically $e = 0$),

$$J^n_{n,i} = \max\!\left(0,\ -\frac{(1+e)\, \mathbf{n}^\top \mathbf{w}^{n,*}_i}{m_i^{-1} + (\mathbf{n} \times \boldsymbol{\rho}^n_i)^2 I_i^{-1}}\right), \tag{16}$$

evaluated only when the contact predicate holds (otherwise $J^n_{n,i} = 0$). With tangent $\mathbf{t} = (1,0)^\top$, a tangential impulse $J^n_{t,i}$ is first computed as if frictionless along $\mathbf{t}$, then Coulomb-capped,

$$J^{n,\text{raw}}_{t,i} = -\frac{\mathbf{t}^\top \mathbf{w}^{n,*}_i}{m_i^{-1} + (\mathbf{t} \times \boldsymbol{\rho}^n_i)^2 I_i^{-1}}, \qquad J^n_{t,i} = \text{clip}\big(J^{n,\text{raw}}_{t,i},\ -\mu J^n_{n,i},\ \mu J^n_{n,i}\big), \tag{17}$$

with friction coefficient μ. The world impulse is $\mathbf{I}^n_i = J^n_{n,i}\mathbf{n} + J^n_{t,i}\mathbf{t}$ and is mapped to rigid-body increments using the lever arm $\widehat{\mathbf{c}}^n_i - \mathbf{x}^n_i$ in the same cross-product formula as for links.

Let ζ be the exponential damping rate and g the gravitational acceleration in the vertical direction. After summing all link and ground impulse contributions into $(\Delta\mathbf{v}^n_i, \Delta\omega^n_i)$, the velocity update is

$$\widetilde{\mathbf{v}}^{\,n+1}_i = e^{-\zeta\Delta t}\, \mathbf{v}^n_i + (0, g\Delta t)^\top + \Delta\mathbf{v}^n_i, \qquad \widetilde{\omega}^{\,n+1}_i = e^{-\zeta\Delta t}\, \omega^n_i + \Delta\omega^n_i, \tag{18}$$

followed by the semi-implicit position update

$$\mathbf{x}^{n+1}_i = \mathbf{x}^n_i + \Delta t\, \widetilde{\mathbf{v}}^{\,n+1}_i, \qquad \theta^{n+1}_i = \theta^n_i + \Delta t\, \widetilde{\omega}^{\,n+1}_i. \tag{19}$$

A.2 Continuous Mechanics

Elastic Mechanics. The mathematical formulation and experimental setups are inspired by Hoyer et al. (2019). We model small-deformation, plane-stress linear elasticity on a rectangular domain $\Omega = (0, L_x) \times (0, L_y)$ discretized by a structured mesh of four-node quadrilateral (Q4) elements with bilinear isoparametric shape functions $N_a(\xi, \eta)$, $a = 1, \ldots, 4$, on the reference cell $(\xi, \eta) \in [-1, 1]^2$. Let $\mathcal{N}$ and $\mathcal{E}$ denote the sets of nodes and elements, and let $\mathbf{u} \in \mathbb{R}^{2|\mathcal{N}|}$ collect nodal displacements (u_x, u_y) in a fixed global ordering. Plane stress relates in-plane strains $\boldsymbol{\varepsilon} = [\varepsilon_{xx}, \varepsilon_{yy}, \gamma_{xy}]^\top$ to stresses $\boldsymbol{\sigma} = [\sigma_{xx}, \sigma_{yy}, \tau_{xy}]^\top$ via $\boldsymbol{\sigma} = \mathbf{D}(E, \nu)\, \boldsymbol{\varepsilon}$, where for isotropic modulus E and Poisson ratio ν,

$$\mathbf{D}(E, \nu) = \frac{E}{1 - \nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix}. \tag{20}$$

For each element e, the strain–displacement matrix $\mathbf{B}_e(\mathbf{x})$ links element nodal displacements $\mathbf{u}_e \in \mathbb{R}^8$ to strains $\boldsymbol{\varepsilon}(\mathbf{x}) = \mathbf{B}_e(\mathbf{x})\mathbf{u}_e$. The element stiffness with uniform Young's modulus E is

$$\mathbf{K}_e(E) = \int_{\Omega_e} \mathbf{B}_e^\top \, \mathbf{D}(E,\nu) \, \mathbf{B}_e \, \mathrm{d}A, \tag{21}$$

evaluated numerically with a 2×2 Gauss quadrature rule on Ω_e. The global stiffness assembles contributions from all elements, $\mathbf{K}(\{E_e\}) = \sum_e \mathbf{L}_e^\top \mathbf{K}_e(E_e)\mathbf{L}_e$, where $\mathbf{L}_e$ maps global to local DOFs.

Each element carries a scalar design density $\bar{\rho}_e \in [0,1]$. The solid isotropic material with penalization (SIMP) law assigns an effective Young's modulus

$$E_e(\bar{\rho}_e) = E_{\min} + \bar{\rho}_e^{\,p}\left(E_0 - E_{\min}\right), \qquad E_{\min} = \varepsilon \, E_0, \tag{22}$$

with reference modulus E_0, penalty exponent $p > 1$, and a small positive stiffness floor $\varepsilon \ll 1$ so that void-like regions retain a well-posed, strictly positive-definite $\mathbf{K}$ on the free subspace.

Static equilibrium with a prescribed nodal force vector $\mathbf{f} \in \mathbb{R}^{2|\mathcal{N}|}$ and homogeneous Dirichlet data on a specified subset of displacement components (fixed DOFs) yields the reduced linear system

$$\mathbf{K}_{ff} \, \mathbf{u}_f = \mathbf{f}_f, \tag{23}$$

where subscripts f denote rows and columns associated with unconstrained DOFs after partitioning. The compliance of the structure for the given load is the elastic strain energy

$$C \;=\; \mathbf{f}^\top \mathbf{u} \;=\; \mathbf{u}_f^\top \mathbf{K}_{ff} \, \mathbf{u}_f \;=\; \int_\Omega \boldsymbol{\sigma}^\top \boldsymbol{\varepsilon} \, \mathrm{d}A, \tag{24}$$

evaluated consistently with the discrete Q4 approximation above.

Fluid Mechanics. We consider a two-dimensional sectional wind tunnel on a rectangle $\Omega = (0, L_x) \times (0, H)$ with coordinates (x, y), where $y = 0$ is the floor and $y = H$ the ceiling. A parameterized car silhouette is described by a single smooth graph $y = y_{\mathrm{body}}(x)$ for $x \in (0, L_x)$. In the implementation, the upper boundary is the sum of a fixed deck height (proportional to H), a Gaussian bump in x controlled by amplitude and width parameters, an algebraic rear taper proportional to $\left(1 - x/L_x\right)_+^2$, followed by a hard cap so that $y_{\mathrm{body}} \le c\,H$ with $c < 1$. The solid region used by the immersed-body mask lies below this surface, $\{(x, y) : 0 < y < y_{\mathrm{body}}(x)\}$, smoothed across a thin transition layer. The velocity $\mathbf{u} = (u, v)^\top$, pressure p, and constant density ρ satisfy a Brinkman–Stokes system posed on the entire tunnel rectangle,

$$-\mu \, \nabla^2 \mathbf{u} + \nabla p + \eta \, \chi \, \mathbf{u} = \mathbf{0}, \qquad \nabla \cdot \mathbf{u} = 0,$$

where μ is the dynamic viscosity and $\chi(\mathbf{x}) \in [0, 1]$ is a smoothed indicator with $\chi \approx 1$ in the car region and $\chi \approx 0$ in the exterior fluid, built from a logistic mollification of $y_{\mathrm{body}}(x) - y$ with width $\tau > 0$. The penalization coefficient η is taken mesh-dependent, scaling like $\mu/\Delta x^2$ for a cell size Δx comparable to L_x/n_x, so that explicit pseudo-time stepping remains stable while enforcing $\mathbf{u} \approx \mathbf{0}$ where $\chi \approx 1$. Where $\chi \approx 0$, the equations reduce to incompressible Stokes in the exterior fluid.

A uniform streamwise inlet $u = U_\infty$, $v = 0$ is imposed at $x = 0$, no-slip $\mathbf{u} = \mathbf{0}$ on the top and bottom walls, and an approximate outflow at $x = L_x$ by extrapolating (u, v) from the penultimate column (zero normal gradient of velocity). In the discrete implementation, the no-slip row at $y = 0$ is applied after the inlet column, so the bottom-left corner is no-slip rather than plug-flow; this affects only a single grid point at the resolution used here. Because the model is marched explicitly in a pseudo-time τ_{ac}, an artificial compressibility pressure update is used,

$$\frac{\partial p}{\partial \tau_{\mathrm{ac}}} + \beta^2 \, \nabla \cdot \mathbf{u} = 0,$$

with β chosen proportional to U_∞ in the code. The velocity is advanced with an explicit Euler step combining viscous diffusion, pressure gradients scaled by ρ^{-1}, and the Brinkman sink $-(\eta/\rho)\chi\mathbf{u}$; the pressure is then projected to zero mean. Only a finite number of pseudo-time iterations is taken, so the reported fields approximate a steady discrete Brinkman–Stokes–AC fixed point at the chosen resolution rather than a fully converged continuum limit.

A streamwise drag force per unit span is estimated from the relaxed discrete fields by a two-term volume integral that contracts p and a single velocity gradient component with gradients of χ,

$$D \;\approx\; \rho\int_{\Omega} p\,\frac{\partial\chi}{\partial x}\,\mathrm{d}A \;-\; \mu\int_{\Omega}\frac{\partial u}{\partial y}\,\frac{\partial\chi}{\partial y}\,\mathrm{d}A,$$

with an overall sign convention fixed so that $D > 0$ corresponds to the resistive streamwise force used in optimization. The pressure contribution is in the spirit of Peskin-type IBM force density $-\nabla\chi$ contracted with the pressure; the second term is a reduced viscous proxy (it retains only $\partial u/\partial y$ rather than the full symmetric viscous traction on the interface) and should be read as an engineering surrogate at low mesh Reynolds number, not a high-fidelity boundary traction integral.

The surface $y_{\mathrm{body}}(x)$ is controlled by a latent vector $\mathbf{z} \in \mathbb{R}^4$ mapped through bounded nonlinearities to a small set of shape scales (bump amplitude, longitudinal width, streamwise center, and a signed rear taper). This enforces a simple single-hump family suitable for few-parameter inverse design.

A.3 Atomistic Simulations

Density Functional Theory. Kohn-Sham Density Functional Theory (KS-DFT) is a foundational computational framework that simplifies the complex, many-body interactions of electrons by mapping them onto a system of non-interacting particles, allowing researchers to predict the properties of materials based on their overall electron density rather than tracking every individual electron. We implement a radial restricted "helium-like" Kohn-Sham DFT simulator, i.e., a spherical closed-shell density $\rho(\mathbf{r}) = \rho(r)$ on $r \in [0, R]$ with a single nuclear center. The reduced radial amplitude $u(r)$ (with $u(0) = u(R) = 0$) defines a restricted two-electron mean field via $\rho(r) = \frac{N_e/2}{2\pi r^2}u(r)^2$ and $\int 4\pi r^2\rho\,\mathrm{d}r = N_e$ (intended for N_e=2, i.e. one doubly occupied s-like spatial orbital). The non-interacting kinetic energy uses $-\frac{1}{2}\,\mathrm{d}^2/\mathrm{d}r^2$ on u (uniform radial mesh in the implementation). The external potential is a softened Coulomb $V_{\mathrm{ext}} = -Z/\sqrt{r^2+\delta^2}$; V_H solves the spherical Poisson equation for ρ; V_{xc} comes from a local $\varepsilon_{\mathrm{xc}}(\rho)$ (Slater exchange plus an analytic correlation surrogate). For Kohn-Sham self-consistency, $V_{\mathrm{eff}} = V_{\mathrm{ext}} + V_H + V_{\mathrm{xc}}$ and iterating between ρ and u with linear mixing are used. The total energy is $E = T_s + E_{\mathrm{ext}} + \frac{1}{2}E_H + E_{\mathrm{xc}}$ with the same $4\pi r^2\,\mathrm{d}r$ measure.

Tight Binding. The Tight-Binding (TB) approximation replaces the full single-particle Hamiltonian by a sparse matrix in a minimal local basis: one orthogonal orbital per site, nearest-neighbor hopping only, and no explicit spin degrees of freedom unless stated otherwise. We consider a single spinless orbital on each of $N \geq 3$ sites arranged on a ring (one spatial dimension, nearest neighbors only). Onsite energies ε_n and real hopping magnitudes $t_{n,n+1} > 0$ enter the Hermitian Hamiltonian with matrix elements

$$H_{nn} = \varepsilon_n, \qquad H_{n,n+1} = H_{n+1,n} = -\,t_{n,n+1} \qquad (n = 0, \ldots, N-2), \tag{25}$$

and a closing bond between sites $N-1$ and 0 with the same convention, $H_{N-1,0} = H_{0,N-1} = -\,t_{N-1,0}$ in the purely periodic problem. Real t and this sign choice correspond to zero magnetic flux through the ring in the usual tight-binding sense (no Peierls phases on the intracell bonds).

To embed this N-site cluster as one supercell of an infinite 1D periodic chain, let a denote the nearest-neighbor distance along the backbone and $A = Na$ the supercell length. A crystal wavevector k enters only through the intercell link that reconnects site $N-1$ to site 0 of the next cell. The Bloch-reduced $N \times N$ Hamiltonian $H(k)$ keeps the same intracell matrix elements as above, while the wrap becomes

$$H_{N-1,0}(k) = -\,t_{N-1,0}\,e^{ikA}, \qquad H_{0,N-1}(k) = -\,t_{N-1,0}\,e^{-ikA}, \tag{26}$$

so $H(k)$ is Hermitian for real k, and $H(k+2\pi/A) = H(k)$. For each k, the single-particle energies are the eigenvalues of $H(k)$; they may be sorted ascending at fixed k for plotting. The finite supercell spectrum used for total energies and eigenvectors at the supercell origin is $H(0)$, i.e. the same ring Hamiltonian without a phase on the closing bond. The total density of states per site is the spectral measure of $H(0)$,

$$D(E) = \frac{1}{N}\sum_{j=1}^{N}\delta\big(E - E_j\big), \tag{27}$$

with $\{E_j\}$ the eigenvalues of $H(0)$. In the benchmarked parameter-identification task, the loss matches $\{E_j\}$ and the Bloch band matrix $E(k)$ to a frozen reference (weights 1 and 0.25); a Gaussian-smoothed DOS is available from the same spectrum for plotting only.

A.4 RENDERING

Gaussian Splatting. We consider a fixed image grid of size $H \times W$ over a normalized square domain $\mathcal{D} = [-1, 1]^2$. Each pixel center is denoted $\mathbf{u}_{ij} \in \mathcal{D}$ for $i \in \{1, \dots, H\}$, $j \in \{1, \dots, W\}$. The renderer places K splats. Splat k is specified by a center $\mathbf{x}_k \in \mathbb{R}^2$, a positive scale σ_k, an opacity $\alpha_k \in (0, 1)$, and a diffuse color $\mathbf{c}_k \in (0, 1)^3$. Let $r_k^2(\mathbf{u}) = \|\mathbf{u} - \mathbf{x}_k\|^2$. The splat weight on the image plane is

$$g_k(\mathbf{u}) = \exp\left(-\frac{r_k^2(\mathbf{u})}{2\sigma_k^2}\right), \qquad w_k(\mathbf{u}) = \alpha_k \, g_k(\mathbf{u}). \tag{28}$$

Thus w_k is non-negative and peaks at $\mathbf{x}_k$ with characteristic width σ_k. In the implementation, σ_k is obtained from a unconstrained parameter via a smooth positive map (softplus) plus a small floor, so that σ_k stays strictly positive and gradients remain stable.

At each $\mathbf{u}$, splat colors are combined with a normalized weighted average of the diffuse colors:

$$\widehat{\mathbf{c}}(\mathbf{u}) = \frac{\sum_{k=1}^{K} w_k(\mathbf{u}) \, \mathbf{c}_k}{\sum_{k=1}^{K} w_k(\mathbf{u}) + \varepsilon}, \tag{29}$$

with a fixed small $\varepsilon > 0$ only to avoid division by zero in regions where all weights vanish. When $\sum_k w_k(\mathbf{u}) > 0$, $\widehat{\mathbf{c}}(\mathbf{u})$ lies in the convex hull of the $\mathbf{c}_k$ because the w_k are non-negative. Splats are blended in an orderless way: no explicit back-to-front sort is required. Let $S(\mathbf{u}) = \sum_{k=1}^{K} w_k(\mathbf{u})$ act as a non-negative accumulated opacity density at $\mathbf{u}$. A smooth coverage is

$$A(\mathbf{u}) = 1 - \mathrm{e}^{-S(\mathbf{u})} \in [0, 1). \tag{30}$$

For large S, $A \to 1$; for $S \to 0$, $A \sim S$. Given a constant background color $\mathbf{I}_{\mathrm{bg}}$ (scalar gray or RGB), the displayed image is

$$\mathbf{I}(\mathbf{u}) = A(\mathbf{u}) \, \widehat{\mathbf{c}}(\mathbf{u}) + \big(1 - A(\mathbf{u})\big) \, \mathbf{I}_{\mathrm{bg}}. \tag{31}$$

The continuous fields g_k, w_k, S, A, $\widehat{\mathbf{c}}$, and $\mathbf{I}$ are evaluated on the regular grid $\{\mathbf{u}_{ij}\}$ only. Coordinates $\mathbf{u}_{ij}$ are generated by uniform sampling of $[-1, 1]$ along each axis (tensor grid), so pixel spacing is uniform in the normalized frame. All operations (Gaussian evaluation, sums, exponentials, and the final blend) are applied pointwise on that grid; the result is an $H \times W \times 3$ RGB image in $[0, 1]^3$ suitable as a differentiable forward map in autodiff frameworks.

Atmospheric Scattering. Sunlight entering Earth's atmosphere interacts with air molecules and aerosols through scattering and absorption Scattering redirects photons out of the incident beam; absorption removes energy from the beam and heats the medium. Because molecular (Rayleigh) scattering scales strongly with wavelength, short wavelengths are scattered more efficiently than long ones, which is why clear skies appear blue and sunsets appear reddened when long optical paths sample more scattering and extinction. Larger particles (haze, cloud droplets) contribute Mie scattering, which is more forward-peaked and less spectrally selective, producing whitish hazes and bright glints around the sun. For rendering and inverse problems, the full radiative transfer equation couples emission, absorption, and multiple scattering; practical sky models often approximate the visible sky by single scattering along a view ray plus analytic or tabulated optical depths, trading physical completeness for speed and differentiability while preserving the main perceptual cues (gradient, polarization of color with elevation, solar disk glow).

The scene is a concentric-shell atmosphere between planet radius R_g and outer radius R_t in a planet-centered frame. Pinhole rays $(\mathbf{o}, \hat{\mathbf{d}})$ are intersected with the outer sphere; integration runs from entry to exit along $\hat{\mathbf{d}}$, truncated at the first intersection with the ground sphere of radius R_g so samples stay above the surface. At a point $\mathbf{p}$ with altitude $h = \max(\|\mathbf{p}\| - R_g, 0)$, scalar number densities follow exponential atmospheres $\rho_R(h) = \exp(-h/H_R)$ and $\rho_M(h) = \exp(-h/H_M)$, with scale heights

H_R, H_M. Per-channel extinction uses RGB coefficients $\boldsymbol{\beta}_R, \boldsymbol{\beta}_M$ (1/m) and an optional absorption vector $\boldsymbol{\beta}_A$ weighted by the same ρ_R factor:

$$\boldsymbol{\sigma}_t(\mathbf{p}) = \rho_R(h)\,\boldsymbol{\beta}_R + \rho_M(h)\,\boldsymbol{\beta}_M + \rho_R(h)\,\boldsymbol{\beta}_A. \tag{32}$$

Along the view ray, samples are spaced uniformly in distance; a cumulative sum approximates $\boldsymbol{\tau}_{\text{view}}(s) = \int_{s_{\min}}^{s} \boldsymbol{\sigma}_t(\mathbf{o} + s'\hat{\mathbf{d}})\,\mathrm{d}s'$. From each sample $\mathbf{p}$, a second quadrature marches toward the (unit) sun direction $\hat{\mathbf{s}}$ out to the atmosphere shell, yielding $\boldsymbol{\tau}_{\odot}(\mathbf{p})$ for sunlight attenuation to $\mathbf{p}$. Let $\mu = \hat{\mathbf{d}}\cdot\hat{\mathbf{s}}$. Rayleigh uses $P_R(\mu) = \frac{3}{16\pi}(1+\mu^2)$; Mie uses the Henyey–Greenstein phase $P_M(\mu \mid g)$. The in-scattered radiance accumulated along the camera ray is a discrete sum of

$$\Big(\rho_R\,\boldsymbol{\beta}_R P_R(\mu) + \rho_M\,\boldsymbol{\beta}_M P_M(\mu \mid g)\Big)\,\exp\big(-(\boldsymbol{\tau}_{\text{view}} + \boldsymbol{\tau}_{\odot})\big)\,\Delta s, \tag{33}$$

scaled by a per-channel sun intensity $\mathbf{I}_{\odot}$, exposure, optional white-balance normalization, and mapped to $[0,1]$ by $\mathbf{L} \mapsto 1 - \exp(-\mathbf{L})$ (invalid rays return black).

A.5 Semi-Empirical Physics Models

Tire Magic Formula. The magic formula is a semi-empirical model for tire dynamics which describes the relationship between the slip ratio and slip angle of a tire and the resulting longitudinal and lateral forces. The model is widely used in vehicle dynamics simulations and control systems. The core equation of the magic formula can be expressed as follows:

$$y = D\sin(C\arctan(Bx - E(Bx - \arctan(Bx)))). \tag{34}$$

It is used to model the longitudinal force as a function of the slip ratio, in which case x is the slip ratio κ, and y is the longitudinal force F_x. It is also used to model the lateral force as a function of the slip angle, in which case x is the slip angle α, and y is the lateral force F_y. The parameters B, C, D, and E can be tuned to fit experimental data, allowing the magic formula to accurately capture the complex behavior of tire forces under various conditions.

Battery Thevenin Model. The Thevenin model simplifies the complex electrochemical processes in a battery into an equivalent circuit consisting of a voltage source and resistors. We use a first-order Thevenin model with one RC polarization branch. Let $I(t)$ denote the applied current at time t (positive for discharge in the usual sign convention; the benchmark uses bipolar random pulse profiles). Let Q denote the nominal capacity and $\eta \in (0,1]$ denote the coulombic efficiency. Let R_0 denote ohmic resistance, and (R_1, C_1) characterize polarization dynamics. The model state is the state of charge $\mathrm{SOC}(t)$ and the RC polarization voltage $V_{\mathrm{rc}}(t)$:

$$\dot{\mathrm{SOC}}(t) = -\,\eta\,\frac{I(t)}{Q}, \tag{35}$$

$$\dot{V}_{\mathrm{rc}}(t) = -\frac{1}{R_1 C_1} V_{\mathrm{rc}}(t) + \frac{1}{C_1} I(t), \tag{36}$$

and the terminal voltage is

$$V(t) = V_{\mathrm{oc}}(\mathrm{SOC}(t)) - R_0 I(t) - V_{\mathrm{rc}}(t). \tag{37}$$

The open-circuit voltage is parameterized as a smooth SoC map; in our implementation we use a cubic polynomial

$$V_{\mathrm{oc}}(s) = a_0 + a_1 s + a_2 s^2 + a_3 s^3, \qquad s \in [0,1]. \tag{38}$$

For simulation, this continuous-time model is discretized at step size Δt and integrated in a differentiable manner, with SoC constrained to $[0,1]$. In the parameter-identification task, both the circuit tuple (Q, η, R_0, R_1, C_1) and the OCV coefficients $(a_0, \ldots, a_3)$ are learnable; the objective is $\mathrm{MSE}(V) + 0.2\,\mathrm{MSE}(\mathrm{SOC})$ against trajectories from a frozen ground-truth model driven by the same currents and initial SoC.

B Details of Inverse Problems

This appendix states the design variables and scalar objectives of each benchmarked inverse problem. The notations follow Appendix A.

B.1 Parameter Identification

DFT XC Functional Fitting. We fit two positive LDA exchange–correlation scales of the helium-like radial Kohn–Sham simulator in Appendix A. The learnable parameters are unconstrained logs $\ell_x, \ell_c \in \mathbb{R}$, mapped by $c_x = \mathrm{e}^{\ell_x}$ and $c_c = \mathrm{e}^{\ell_c}$ onto the exchange and correlation multipliers of the local $\varepsilon_{\mathrm{xc}}$. A frozen reference run with $(c_x, c_c) = (1, 1)$ produces target total energy E_{ref} and radial density $\rho_{\mathrm{ref}}(r)$ on the same mesh. With the learnable scales, one SCF forward pass returns (E, ρ), and the objective is

$$\mathcal{L} \;=\; \big(E - E_{\mathrm{ref}}\big)^2 \;+\; 2\int_0^R 4\pi r^2 \big(\rho(r) - \rho_{\mathrm{ref}}(r)\big)^2 \,\mathrm{d}r, \tag{39}$$

where the density integral uses the same spherical quadrature as the energy evaluation. Defaults use $Z = N_e = 2$, $n_{\mathrm{radial}} = 257$ radial points, 28 SCF iterations with mixing 0.35, and softened Coulomb parameter $\delta = 0.08$.

Tire Parameter Fitting. Longitudinal and lateral tire forces are modeled by independent Magic Formula copies with parameter tuples (B_x, C_x, D_x, E_x) and (B_y, C_y, D_y, E_y) (eight scalars in total). A frozen ground-truth tuple generates targets $F_x^\star(\kappa)$ and $F_y^\star(\alpha)$ on $n = 256$ random slip samples $\kappa \sim \mathcal{U}(-0.2, 0.2)$ and $\alpha \sim \mathcal{U}(-0.5, 0.5)$. With predicted forces (F_x, F_y) from the learnable parameters, the objective is

$$\mathcal{L} \;=\; \mathrm{mean}\big((F_x - F_x^\star)^{\odot 2}\big) \;+\; \mathrm{mean}\big((F_y - F_y^\star)^{\odot 2}\big). \tag{40}$$

Battery Parameter Fitting. The learnable variables are the Thevenin circuit tuple (Q, η, R_0, R_1, C_1) together with cubic OCV coefficients (a_0, a_1, a_2, a_3) from Appendix A (nine scalars). A frozen ground-truth model is driven by the same random bipolar pulse currents and initial SoC values over $n_{\mathrm{seq}} = 32$ sequences of horizon $T = 80$ with $\Delta t = 1$. Writing $V_t^{(s)}$ and $\mathrm{SOC}_t^{(s)}$ for predicted trajectories and starred symbols for targets, the objective is

$$\mathcal{L} \;=\; \mathrm{mean}_{s,t}\big((V_t^{(s)} - V_t^{(s),\star})^2\big) \;+\; 0.2\,\mathrm{mean}_{s,t}\big((\mathrm{SOC}_t^{(s)} - \mathrm{SOC}_t^{(s),\star})^2\big). \tag{41}$$

Tight Binding Parameter Fitting. On an $N = 16$ site ring, the learnable variables are the on-site energies $\{\varepsilon_n\}_{n=0}^{N-1}$ and nearest-neighbor hoppings $\{t_{n,n+1}\}$ (32 scalars), initialized by additive Gaussian noise around a frozen reference ($\varepsilon_n \leftarrow \varepsilon_n^\star + 0.35\,\xi_n$, $t_{n,n+1} \leftarrow t_{n,n+1}^\star + 0.25\,\zeta_n$ with $\xi_n, \zeta_n \sim \mathcal{N}(0,1)$). Let $\{E_j\}$ be the eigenvalues of the supercell Hamiltonian $H(0)$ and $E(k)$ the Bloch band matrix on 48 crystal momenta. Against frozen reference spectra $\{E_j^\star\}$ and $E^\star(k)$, the objective is

$$\mathcal{L} \;=\; \mathrm{mean}_j\big((E_j - E_j^\star)^2\big) \;+\; 0.25\,\mathrm{mean}_{k,j}\big((E_j(k) - E_j^\star(k))^2\big). \tag{42}$$

A Gaussian-smoothed density of states is computed for visualization only and does not enter $\mathcal{L}$.

Gaussian Splatting 2D Reconstruction. We fit $K = 32$ splat parameters on an $H = W = 48$ image grid: centers $\mathbf{x}_k$, positive scales σ_k (via a softplus map), opacities α_k, and RGB colors $\mathbf{c}_k$, for 224 design variables in total. Let $\mathbf{I}_\theta$ denote the rendered image under Appendix A and $\mathbf{I}^\star$ a fixed target image (bundled reference). The objective is the pixelwise mean squared error

$$\mathcal{L} \;=\; \mathrm{mean}\big((\mathbf{I}_\theta - \mathbf{I}^\star)^{\odot 2}\big). \tag{43}$$

B.2 Inverse Design

Projectile Range Maximization. With fixed $h_0 = v_0 = g = 9.8$, the sole design variable is the launch angle θ. The simulator returns the closed-form horizontal range $s_x(\theta)$ of Appendix A. The benchmark minimizes the negated range,

$$\mathcal{L}(\theta) \;=\; -\,s_x(\theta), \tag{44}$$

so that standard first-order and global minimizers maximize flight distance.

Fluid Shape Optimization. The car silhouette is controlled by a latent vector $\mathbf{z} \in \mathbb{R}^4$ mapped to bump amplitude, width, center, and rear taper as in Appendix A. Tunnel defaults are $L_x = 2$, $H = 1$, $\rho = 1$, $\mu = 0.08$, $U_\infty = 1$, on a 20×10 grid with 120 artificial-compressibility pseudo-steps. The objective is the scalar streamwise drag estimate $D(\mathbf{z})$,

$$\mathcal{L}(\mathbf{z}) \;=\; D(\mathbf{z}). \tag{45}$$

Atmosphere Chromaticity Maximization. A pinhole camera renders an $H \times W = 28 \times 40$ sky image $\mathbf{I} \in \mathbb{R}^{H\times W\times 3}$ under Appendix A. Learnable variables are an unconstrained sun direction $\mathbf{s}_{\text{raw}} \in \mathbb{R}^3$ (normalized to $\hat{\mathbf{s}}$), scale heights (H_R, H_M) via logs, RGB extinction vectors $(\boldsymbol{\beta}_R, \boldsymbol{\beta}_M, \boldsymbol{\beta}_A)$ via logs, and Mie asymmetry $g = \sigma(g_{\text{logit}})$, for 15 design variables in total; sun intensity and camera pose are fixed. Let $\boldsymbol{\mu}_{\text{top}} = \text{mean}_j \mathbf{I}_{0j}$ and $\boldsymbol{\mu}_{\text{bot}} = \text{mean}_j \mathbf{I}_{H-1,j}$ denote the mean RGB of the top and bottom rows. Vertical chromatic contrast is

$$c \;=\; \text{mean}\big(|\boldsymbol{\mu}_{\text{top}} - \boldsymbol{\mu}_{\text{bot}}|\big), \tag{46}$$

and the minimized objective is $\mathcal{L} = -c$.

Structural Topology Optimization. On a half MBB beam discretized by a 30×20 Q4 mesh (600 element densities), SIMP moduli follow Appendix A with $E_0 = 1$, $\nu = 0.3$, penalty $p = 3$, and $E_{\min}/E_0 = 10^{-5}$ on a domain of size $(L_x, L_y) = (2, 1)$. Let $C(\boldsymbol{\rho})$ be compliance, C_{ref} the compliance of the all-solid field $\boldsymbol{\rho} \equiv 1$ (clamped below by 10^{-12}), and $\overline{\rho} = \text{mean}_e \rho_e$ the mean density with target volume fraction $\overline{\rho}^\star = 0.5$. The objective is

$$\mathcal{L}(\boldsymbol{\rho}) \;=\; \frac{C(\boldsymbol{\rho})}{C_{\text{ref}}} \;+\; 25\,\big(\overline{\rho} - \overline{\rho}^\star\big)^2. \tag{47}$$

B.3 Optimal Control

Rigid-Body Robot Setpoint. We optimize an open-loop actuation table $\mathbf{U} \in \mathbb{R}^{T\times n_{\text{s}}}$ with $T = 10$ and $n_{\text{s}} = 7$ (Difftaichi robotA springs), applied as $u_e^n = \tanh(U_e^n)$ with $\Delta t = 10^{-3}$. Let $\mathbf{p}_T, \mathbf{v}_T \in \mathbb{R}^2$ be the terminal center-of-mass position and velocity of the designated head body, and $\mathbf{g} = (0.9, 0.15)^\top$ the planar goal. The objective is

$$\begin{aligned}\mathcal{L} = \text{mean}\big((\mathbf{p}_T - \mathbf{g})^{\odot 2}\big) + 0.12\,\text{mean}\big(\mathbf{v}_T^{\odot 2}\big) \\ + 10^{-3}\,\text{mean}\big(\mathbf{U}^{\odot 2}\big) + 10^{-3}\,\text{mean}\big((\mathbf{U}_{t+1} - \mathbf{U}_t)^{\odot 2}\big).\end{aligned} \tag{48}$$

Flight Dynamics Range Maximization. An open-loop logit table $\mathbf{u} \in \mathbb{R}^{T\times 2}$ with $T = 200$ and $\Delta t = 1$ produces aerodynamic coefficients

$$C_L^t = 0.65\,\tanh(u_0^t), \qquad C_D^t = 0.04 + 0.22\,\text{sigmoid}(u_1^t), \tag{49}$$

integrated from the default initial state of Appendix A. Writing x_T for the terminal ground-range coordinate (meters) and $U = 10^6$ m,

$$\begin{aligned}\mathcal{L} = -\frac{x_T}{U} + \frac{2\times 10^{-5}}{U}\,\text{mean}\big(\mathbf{u}^{\odot 2}\big) \\ + \frac{1\times 10^{-5}}{U}\,\text{mean}\big((\mathbf{u}_{t+1} - \mathbf{u}_t)^{\odot 2}\big).\end{aligned} \tag{50}$$

Mass-Spring Robot Reach. Analogously, we optimize $\mathbf{U} \in \mathbb{R}^{T\times n_{\text{s}}}$ with $T = 100$ and $n_{\text{s}} = 11$ (Difftaichi mass–spring robotA), applied as $u_e^n = \tanh(U_e^n)$ with $\Delta t = 4 \times 10^{-3}$. With head position/velocity $(\mathbf{p}_T, \mathbf{v}_T)$ and goal $\mathbf{g} = (0.9, 0.2)^\top$,

$$\begin{aligned}\mathcal{L} = \text{mean}\big((\mathbf{p}_T - \mathbf{g})^{\odot 2}\big) + 0.15\,\text{mean}\big(\mathbf{v}_T^{\odot 2}\big) \\ + 2\times 10^{-4}\,\text{mean}\big(\mathbf{U}^{\odot 2}\big) + 2\times 10^{-4}\,\text{mean}\big((\mathbf{U}_{t+1} - \mathbf{U}_t)^{\odot 2}\big).\end{aligned} \tag{51}$$

Table 7: Optimizer hyper-parameter search space for the benchmark.

Name	Searched / fixed configurations
SGD	search `lr`, `momentum` (8 configs)
RMSprop	search `lr`, `momentum` (8 configs)
Adam	search `lr`, `beta1`; fixed `beta2=0.999` (8 configs)
AdamW	search `lr`, `beta1`; fixed `beta2=0.999` (8 configs)
LBFGS	search `lr`; fixed `max_iter=10`, `line_search_fn=strong_wolfe` (4 configs)
KFAC	search `lr`, `damping`; fixed `momentum=0.9`, `beta=0.95`, `precondition_frequency=10` (12 configs)
Shampoo	search `lr`, `damping`; fixed `beta=0.95`, `precondition_frequency=10` (12 configs)
SOAP	search `lr`, `damping`; fixed `betas=(0.9,0.99)`, `beta_precond=0.95`, `precondition_frequency=10` (12 configs)
RandomSearch	search `sigma`, `candidates_per_step`, `polish_lr`; fixed `polish_steps=5`, `clamp=10` (36 configs)
SimulatedAnnealing	search `sigma`, `temperature`, `candidates_per_step`; fixed `cooling=0.99`, `polish_steps=5`, `polish_lr=1e-2`, `clamp=10` (27 configs)
ParticleSwarm	search `init_sigma`, `swarm_size`, `polish_lr`; fixed `polish_steps=5`, `clamp=10` (36 configs)
DifferentialEvolution	search `init_sigma`, `population_size`, `polish_lr`; fixed `polish_steps=5`, `clamp=10` (36 configs)

C Details of Experiment Setups and Results

C.1 Experiment Setups

For each task and each of the 12 optimizers, we evaluate a shared hyper-parameter grid (207 configurations in total across optimizers) and select the best configuration by minimizing the mean best loss over random initializations. Each configuration is repeated 3 times, and each run executes $T = 100$ optimizer epochs. One epoch is one call to the optimizer's update rule, with the scalar loss defined by the task's loss function. Experiments are run on an NVIDIA A100 GPU (80 GB). We check the convergence of each run and non-converged repeats are excluded from hyper-parameter selection and from the reported mean/std; selection otherwise follows the same minimize-mean-best-loss rule on the remaining repeats.

Let $\theta^{(0)}$ denote the baseline design-variable tensors, i.e., floating-point entries of `model.named_parameters()` only. Simulator buffers such as FEM stiffness stacks, nodal forces, and mesh coordinates are copied unchanged. For run r, each design variable θ is reinitialized as

$$\tilde{\theta} = \theta + \sigma_{\text{init}} \cdot \max\big(m(\theta),\, 10^{-2}\big) \cdot \xi, \qquad \xi \sim \mathcal{N}(0, I), \tag{52}$$

where $m(\theta)$ is the mean of $|\theta|$, and σ_{init} is the init noise (set to 0.05). Noise samples use a `torch.Generator` with seed 0.

Table 7 lists the searched hyper-parameters and fixed secondary knobs for each optimizer. Candidate values are drawn from shared global grids: `lr`/`polish_lr` $\in \{10^{-3}, 10^{-2}, 10^{-1}, 1\}$, `momentum`/`beta1` $\in \{0, 0.9\}$, `damping` $\in \{10^{-4}, 10^{-3}, 10^{-2}\}$, `sigma`/`init_sigma` $\in \{0.05, 0.1, 0.5\}$, `population` $\in \{8, 16, 32\}$, `temperature` $\in \{0.5, 1, 2\}$. For approximate second-order methods, each outer step uses the autograd gradient from the task's forward–backward pass. For global optimizers, after each global search proposal, the optimizers optionally run a short gradient-descent polish (with `polish_steps=5` fixed) to locally refine the best candidate.

Exceptions inside the inner training loop are caught per run: the failure is logged, and the benchmark continues with the next repeat or optimizer without terminating the entire sweep.

C.2 COMPUTE RESOURCES

We record the wall time and peak GPU memory usage for each run of the selected (best) hyperparameter configuration. The wall time is measured for the full optimization run and converted to a per-step average by dividing by the number of effective epochs; mean/std are computed over the three repeats. Peak GPU memory is likewise averaged over those three best-configuration repeats (reported in MiB as "MB" in the tables).

Table 8: Compute Resource Usage for DFT XC Functional Parameter Fitting.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$1.1 \times 10^{0} \pm 3.1 \times 10^{-2}$	$5.1 \times 10^{1} \pm 0.0$
RMSprop	$1.1 \times 10^{0} \pm 8.3 \times 10^{-3}$	$5.1 \times 10^{1} \pm 0.0$
Adam	$9.6 \times 10^{-1} \pm 1.1 \times 10^{-2}$	$5.1 \times 10^{1} \pm 0.0$
AdamW	$9.6 \times 10^{-1} \pm 1.9 \times 10^{-2}$	$5.1 \times 10^{1} \pm 0.0$
LBFGS	$6.4 \times 10^{-1} \pm 1.6 \times 10^{-2}$	$5.1 \times 10^{1} \pm 1.8 \times 10^{-3}$
KFAC	$1.1 \times 10^{0} \pm 1.2 \times 10^{-2}$	$5.1 \times 10^{1} \pm 0.0$
Shampoo	$1.1 \times 10^{0} \pm 3.3 \times 10^{-3}$	$5.1 \times 10^{1} \pm 0.0$
SOAP	$1.1 \times 10^{0} \pm 5.0 \times 10^{-3}$	$5.1 \times 10^{1} \pm 0.0$
RandomSearch	$1.9 \times 10^{1} \pm 1.6 \times 10^{-1}$	$5.1 \times 10^{1} \pm 0.0$
SimulatedAnnealing	$1.1 \times 10^{1} \pm 3.7 \times 10^{-2}$	$5.1 \times 10^{1} \pm 0.0$
ParticleSwarm	$5.0 \times 10^{0} \pm 1.4 \times 10^{-2}$	$5.1 \times 10^{1} \pm 0.0$
DifferentialEvolution	$5.0 \times 10^{0} \pm 1.8 \times 10^{-2}$	$5.1 \times 10^{1} \pm 0.0$

Table 9: Compute Resource Usage for Tire Magic Formula Parameter Fitting.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$2.2 \times 10^{-3} \pm 8.2 \times 10^{-4}$	$8.7 \times 10^{0} \pm 0.0$
RMSprop	$2.0 \times 10^{-3} \pm 5.0 \times 10^{-5}$	$8.7 \times 10^{0} \pm 0.0$
Adam	$2.4 \times 10^{-3} \pm 1.7 \times 10^{-4}$	$8.7 \times 10^{0} \pm 0.0$
AdamW	$2.6 \times 10^{-3} \pm 2.3 \times 10^{-4}$	$8.7 \times 10^{0} \pm 0.0$
LBFGS	$6.4 \times 10^{-1} \pm 8.4 \times 10^{-2}$	$8.9 \times 10^{0} \pm 9.2 \times 10^{-4}$
KFAC	$2.8 \times 10^{-3} \pm 2.0 \times 10^{-4}$	$8.7 \times 10^{0} \pm 0.0$
Shampoo	$2.6 \times 10^{-3} \pm 4.7 \times 10^{-5}$	$8.7 \times 10^{0} \pm 0.0$
SOAP	$2.6 \times 10^{-3} \pm 9.7 \times 10^{-5}$	$8.7 \times 10^{0} \pm 0.0$
RandomSearch	$7.8 \times 10^{-2} \pm 1.3 \times 10^{-3}$	$8.7 \times 10^{0} \pm 0.0$
SimulatedAnnealing	$1.2 \times 10^{-1} \pm 6.5 \times 10^{-4}$	$8.7 \times 10^{0} \pm 0.0$
ParticleSwarm	$1.0 \times 10^{-1} \pm 2.9 \times 10^{-3}$	$8.7 \times 10^{0} \pm 0.0$
DifferentialEvolution	$1.8 \times 10^{-1} \pm 1.1 \times 10^{-3}$	$8.7 \times 10^{0} \pm 0.0$

Table 10: Compute Resource Usage for Battery Thevenin Model Parameter Fitting.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$9.1 \times 10^{-2} \pm 4.0 \times 10^{-4}$	$9.1 \times 10^{0} \pm 0.0$
RMSprop	$8.6 \times 10^{-2} \pm 1.4 \times 10^{-3}$	$9.1 \times 10^{0} \pm 0.0$
Adam	$8.6 \times 10^{-2} \pm 8.1 \times 10^{-4}$	$9.1 \times 10^{0} \pm 0.0$
AdamW	$8.7 \times 10^{-2} \pm 7.8 \times 10^{-4}$	$9.1 \times 10^{0} \pm 0.0$
LBFGS	$6.4 \times 10^{-1} \pm 2.4 \times 10^{-2}$	$9.2 \times 10^{0} \pm 6.3 \times 10^{-3}$
KFAC	$9.3 \times 10^{-2} \pm 1.7 \times 10^{-3}$	$9.1 \times 10^{0} \pm 0.0$
Shampoo	$8.8 \times 10^{-2} \pm 4.9 \times 10^{-4}$	$9.1 \times 10^{0} \pm 0.0$
SOAP	$8.7 \times 10^{-2} \pm 1.2 \times 10^{-3}$	$9.1 \times 10^{0} \pm 0.0$
RandomSearch	$8.1 \times 10^{-1} \pm 2.0 \times 10^{-2}$	$9.1 \times 10^{0} \pm 0.0$
SimulatedAnnealing	$8.7 \times 10^{-1} \pm 1.7 \times 10^{-2}$	$9.1 \times 10^{0} \pm 0.0$
ParticleSwarm	$8.0 \times 10^{-1} \pm 1.5 \times 10^{-2}$	$9.1 \times 10^{0} \pm 0.0$
DifferentialEvolution	$1.5 \times 10^{0} \pm 6.6 \times 10^{-3}$	$9.1 \times 10^{0} \pm 0.0$

Table 11: Compute Resource Usage for Tight Binding Parameter Fitting.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$6.8 \times 10^{-1} \pm 1.6 \times 10^{-3}$	$1.8 \times 10^{1} \pm 0.0$
RMSprop	$6.9 \times 10^{-1} \pm 9.0 \times 10^{-3}$	$1.8 \times 10^{1} \pm 0.0$
Adam	$8.1 \times 10^{-1} \pm 9.8 \times 10^{-2}$	$1.8 \times 10^{1} \pm 0.0$
AdamW	$1.2 \times 10^{0} \pm 5.5 \times 10^{-3}$	$1.8 \times 10^{1} \pm 0.0$
LBFGS	$2.0 \times 10^{0} \pm 2.1 \times 10^{-2}$	$1.8 \times 10^{1} \pm 4.8 \times 10^{-3}$
KFAC	$1.1 \times 10^{0} \pm 5.7 \times 10^{-3}$	$1.8 \times 10^{1} \pm 0.0$
Shampoo	$1.2 \times 10^{0} \pm 7.1 \times 10^{-3}$	$1.8 \times 10^{1} \pm 0.0$
SOAP	$1.2 \times 10^{0} \pm 5.8 \times 10^{-3}$	$1.8 \times 10^{1} \pm 0.0$
RandomSearch	$1.1 \times 10^{1} \pm 2.8 \times 10^{-1}$	$1.8 \times 10^{1} \pm 0.0$
SimulatedAnnealing	$5.8 \times 10^{0} \pm 6.5 \times 10^{-3}$	$1.8 \times 10^{1} \pm 0.0$
ParticleSwarm	$1.2 \times 10^{1} \pm 3.5 \times 10^{-2}$	$1.8 \times 10^{1} \pm 0.0$
DifferentialEvolution	$5.6 \times 10^{0} \pm 1.4 \times 10^{-2}$	$1.8 \times 10^{1} \pm 0.0$

Table 12: Compute Resource Usage for Gaussian Splatting 2D Reconstruction.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$6.8 \times 10^{-3} \pm 1.1 \times 10^{-4}$	$1.2 \times 10^{1} \pm 0.0$
RMSprop	$6.8 \times 10^{-3} \pm 7.0 \times 10^{-5}$	$1.2 \times 10^{1} \pm 0.0$
Adam	$7.2 \times 10^{-3} \pm 1.7 \times 10^{-4}$	$1.2 \times 10^{1} \pm 0.0$
AdamW	$7.1 \times 10^{-3} \pm 8.9 \times 10^{-5}$	$1.2 \times 10^{1} \pm 0.0$
LBFGS	$3.2 \times 10^{0} \pm 1.6 \times 10^{-1}$	$1.2 \times 10^{1} \pm 0.0$
KFAC	$2.5 \times 10^{-2} \pm 4.1 \times 10^{-4}$	$1.2 \times 10^{1} \pm 0.0$
Shampoo	$1.4 \times 10^{-2} \pm 1.1 \times 10^{-4}$	$1.2 \times 10^{1} \pm 0.0$
SOAP	$1.5 \times 10^{-2} \pm 4.0 \times 10^{-4}$	$1.2 \times 10^{1} \pm 0.0$
RandomSearch	$1.6 \times 10^{-1} \pm 2.2 \times 10^{-3}$	$1.2 \times 10^{1} \pm 0.0$
SimulatedAnnealing	$1.7 \times 10^{-1} \pm 6.6 \times 10^{-4}$	$1.2 \times 10^{1} \pm 0.0$
ParticleSwarm	$3.9 \times 10^{-1} \pm 9.8 \times 10^{-3}$	$1.2 \times 10^{1} \pm 0.0$
DifferentialEvolution	$2.2 \times 10^{-1} \pm 7.7 \times 10^{-4}$	$1.2 \times 10^{1} \pm 0.0$

Table 13: Compute Resource Usage for Projectile Motion Range Maximization.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$1.9 \times 10^{-3} \pm 1.1 \times 10^{-5}$	$8.7 \times 10^{0} \pm 0.0$
RMSprop	$2.1 \times 10^{-3} \pm 7.6 \times 10^{-6}$	$8.7 \times 10^{0} \pm 0.0$
Adam	$2.3 \times 10^{-3} \pm 4.1 \times 10^{-4}$	$8.7 \times 10^{0} \pm 0.0$
AdamW	$2.0 \times 10^{-3} \pm 2.0 \times 10^{-5}$	$8.7 \times 10^{0} \pm 0.0$
LBFGS	$3.2 \times 10^{-2} \pm 8.5 \times 10^{-3}$	$8.8 \times 10^{0} \pm 7.6 \times 10^{-3}$
KFAC	$4.1 \times 10^{-3} \pm 1.2 \times 10^{-4}$	$8.7 \times 10^{0} \pm 0.0$
Shampoo	$2.0 \times 10^{-3} \pm 4.3 \times 10^{-5}$	$8.7 \times 10^{0} \pm 0.0$
SOAP	$1.9 \times 10^{-3} \pm 7.6 \times 10^{-6}$	$8.7 \times 10^{0} \pm 0.0$
RandomSearch	$2.2 \times 10^{-2} \pm 9.7 \times 10^{-5}$	$8.7 \times 10^{0} \pm 0.0$
SimulatedAnnealing	$3.9 \times 10^{-2} \pm 5.0 \times 10^{-3}$	$8.7 \times 10^{0} \pm 0.0$
ParticleSwarm	$3.0 \times 10^{-2} \pm 2.0 \times 10^{-3}$	$8.7 \times 10^{0} \pm 0.0$
DifferentialEvolution	$3.1 \times 10^{-2} \pm 7.5 \times 10^{-4}$	$8.7 \times 10^{0} \pm 0.0$

Table 14: Compute Resource Usage for Fluid Mechanics Shape Optimization.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$4.9 \times 10^{-1} \pm 1.5 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$
RMSprop	$4.8 \times 10^{-1} \pm 7.0 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
Adam	$5.2 \times 10^{-1} \pm 3.1 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
AdamW	$5.1 \times 10^{-1} \pm 2.5 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
LBFGS	$6.6 \times 10^{-1} \pm 2.8 \times 10^{-2}$	$9.4 \times 10^{0} \pm 5.0 \times 10^{-3}$
KFAC	$5.2 \times 10^{-1} \pm 1.9 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$
Shampoo	$4.8 \times 10^{-1} \pm 4.6 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
SOAP	$5.1 \times 10^{-1} \pm 1.9 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
RandomSearch	$3.6 \times 10^{0} \pm 1.0 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$
SimulatedAnnealing	$6.5 \times 10^{0} \pm 6.2 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
ParticleSwarm	$4.5 \times 10^{0} \pm 4.1 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$
DifferentialEvolution	$4.8 \times 10^{0} \pm 1.8 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$

Table 15: Compute Resource Usage for Atmospheric Scattering Chromaticity Maximization.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$1.5 \times 10^{-2} \pm 3.4 \times 10^{-4}$	$3.8 \times 10^{1} \pm 0.0$
RMSprop	$1.4 \times 10^{-2} \pm 8.5 \times 10^{-5}$	$3.8 \times 10^{1} \pm 0.0$
Adam	$1.4 \times 10^{-2} \pm 7.3 \times 10^{-5}$	$3.8 \times 10^{1} \pm 0.0$
AdamW	$1.4 \times 10^{-2} \pm 2.2 \times 10^{-4}$	$3.8 \times 10^{1} \pm 0.0$
LBFGS	$5.0 \times 10^{-1} \pm 4.1 \times 10^{-2}$	$3.8 \times 10^{1} \pm 4.9 \times 10^{-3}$
KFAC	$3.1 \times 10^{-2} \pm 3.1 \times 10^{-4}$	$3.8 \times 10^{1} \pm 0.0$
Shampoo	$1.6 \times 10^{-2} \pm 1.4 \times 10^{-4}$	$3.8 \times 10^{1} \pm 0.0$
SOAP	$1.6 \times 10^{-2} \pm 2.0 \times 10^{-4}$	$3.8 \times 10^{1} \pm 0.0$
RandomSearch	$2.8 \times 10^{-1} \pm 2.2 \times 10^{-3}$	$3.8 \times 10^{1} \pm 0.0$
SimulatedAnnealing	$4.0 \times 10^{-1} \pm 3.5 \times 10^{-4}$	$3.8 \times 10^{1} \pm 0.0$
ParticleSwarm	$5.0 \times 10^{-1} \pm 3.3 \times 10^{-3}$	$3.8 \times 10^{1} \pm 0.0$
DifferentialEvolution	$8.9 \times 10^{-1} \pm 5.4 \times 10^{-3}$	$3.8 \times 10^{1} \pm 0.0$

Table 16: Compute Resource Usage for Elastic Mechanics Topology Optimization.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$3.6 \times 10^{-1} \pm 5.9 \times 10^{-4}$	$4.4 \times 10^{1} \pm 0.0$
RMSprop	$3.4 \times 10^{-1} \pm 5.6 \times 10^{-4}$	$4.4 \times 10^{1} \pm 0.0$
Adam	$3.4 \times 10^{-1} \pm 1.1 \times 10^{-3}$	$4.4 \times 10^{1} \pm 0.0$
AdamW	$3.4 \times 10^{-1} \pm 1.4 \times 10^{-3}$	$4.4 \times 10^{1} \pm 0.0$
LBFGS	$2.0 \times 10^{0} \pm 1.6 \times 10^{-1}$	$4.4 \times 10^{1} \pm 2.2 \times 10^{-2}$
KFAC	$3.5 \times 10^{-1} \pm 2.1 \times 10^{-3}$	$4.9 \times 10^{1} \pm 0.0$
Shampoo	$3.4 \times 10^{-1} \pm 6.8 \times 10^{-4}$	$4.6 \times 10^{1} \pm 0.0$
SOAP	$3.4 \times 10^{-1} \pm 3.1 \times 10^{-3}$	$4.6 \times 10^{1} \pm 0.0$
RandomSearch	$4.3 \times 10^{0} \pm 7.2 \times 10^{-3}$	$4.4 \times 10^{1} \pm 0.0$
SimulatedAnnealing	$4.3 \times 10^{0} \pm 7.8 \times 10^{-3}$	$4.4 \times 10^{1} \pm 0.0$
ParticleSwarm	$2.9 \times 10^{0} \pm 2.2 \times 10^{-2}$	$4.4 \times 10^{1} \pm 0.0$
DifferentialEvolution	$3.0 \times 10^{0} \pm 5.0 \times 10^{-3}$	$4.4 \times 10^{1} \pm 0.0$

Table 17: Compute Resource Usage for Flight Dynamics Range Maximization.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$1.6 \times 10^{0} \pm 1.4 \times 10^{-2}$	$1.1 \times 10^{1} \pm 0.0$
RMSprop	$1.7 \times 10^{0} \pm 1.1 \times 10^{-2}$	$1.1 \times 10^{1} \pm 0.0$
Adam	$1.6 \times 10^{0} \pm 7.3 \times 10^{-3}$	$1.1 \times 10^{1} \pm 0.0$
AdamW	$1.5 \times 10^{0} \pm 5.0 \times 10^{-3}$	$1.1 \times 10^{1} \pm 0.0$
LBFGS	$5.0 \times 10^{0} \pm 4.9 \times 10^{0}$	$1.1 \times 10^{1} \pm 2.1 \times 10^{-2}$
KFAC	$2.1 \times 10^{0} \pm 5.1 \times 10^{-3}$	$1.3 \times 10^{1} \pm 0.0$
Shampoo	$2.1 \times 10^{0} \pm 1.8 \times 10^{-3}$	$1.1 \times 10^{1} \pm 0.0$
SOAP	$2.1 \times 10^{0} \pm 9.8 \times 10^{-3}$	$1.1 \times 10^{1} \pm 0.0$
RandomSearch	$1.7 \times 10^{1} \pm 6.4 \times 10^{-2}$	$1.1 \times 10^{1} \pm 0.0$
SimulatedAnnealing	$1.7 \times 10^{1} \pm 2.4 \times 10^{-2}$	$1.1 \times 10^{1} \pm 0.0$
ParticleSwarm	$1.1 \times 10^{1} \pm 1.1 \times 10^{-1}$	$1.1 \times 10^{1} \pm 0.0$
DifferentialEvolution	$7.6 \times 10^{0} \pm 7.6 \times 10^{-2}$	$1.1 \times 10^{1} \pm 0.0$

Table 18: Compute Resource Usage for Rigid-Body Robot Setpoint.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$2.9 \times 10^{0} \pm 5.4 \times 10^{-2}$	$1.9 \times 10^{1} \pm 0.0$
RMSprop	$2.8 \times 10^{0} \pm 2.2 \times 10^{-2}$	$1.9 \times 10^{1} \pm 0.0$
Adam	$2.6 \times 10^{0} \pm 3.5 \times 10^{-2}$	$1.9 \times 10^{1} \pm 0.0$
AdamW	$2.8 \times 10^{0} \pm 1.8 \times 10^{-1}$	$1.9 \times 10^{1} \pm 0.0$
LBFGS	$1.3 \times 10^{1} \pm 7.2 \times 10^{0}$	$1.9 \times 10^{1} \pm 1.3 \times 10^{-2}$
KFAC	$3.6 \times 10^{0} \pm 5.2 \times 10^{-3}$	$1.9 \times 10^{1} \pm 0.0$
Shampoo	$3.6 \times 10^{0} \pm 2.7 \times 10^{-2}$	$1.9 \times 10^{1} \pm 0.0$
SOAP	$3.7 \times 10^{0} \pm 5.6 \times 10^{-2}$	$1.9 \times 10^{1} \pm 0.0$
RandomSearch	$6.7 \times 10^{1} \pm 1.3 \times 10^{0}$	$1.9 \times 10^{1} \pm 0.0$
SimulatedAnnealing	$2.8 \times 10^{1} \pm 4.9 \times 10^{-2}$	$1.9 \times 10^{1} \pm 0.0$
ParticleSwarm	$1.4 \times 10^{1} \pm 5.0 \times 10^{-2}$	$1.9 \times 10^{1} \pm 0.0$
DifferentialEvolution	$5.7 \times 10^{0} \pm 1.5 \times 10^{-1}$	$1.9 \times 10^{1} \pm 0.0$

Table 19: Compute Resource Usage for Mass-Spring Robot Reach.

Optimizer	Step wall (s), mean ± std	Peak GPU (MB), mean ± std
SGD	$1.4 \times 10^{0} \pm 1.1 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$
RMSprop	$1.4 \times 10^{0} \pm 2.3 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
Adam	$1.4 \times 10^{0} \pm 3.8 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
AdamW	$1.4 \times 10^{0} \pm 1.5 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$
LBFGS	$3.9 \times 10^{0} \pm 6.5 \times 10^{-1}$	$9.4 \times 10^{0} \pm 2.3 \times 10^{-2}$
KFAC	$7.5 \times 10^{-1} \pm 2.0 \times 10^{-3}$	$1.1 \times 10^{1} \pm 0.0$
Shampoo	$7.5 \times 10^{-1} \pm 4.0 \times 10^{-3}$	$9.3 \times 10^{0} \pm 0.0$
SOAP	$7.5 \times 10^{-1} \pm 9.1 \times 10^{-4}$	$9.4 \times 10^{0} \pm 0.0$
RandomSearch	$1.6 \times 10^{1} \pm 4.7 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$
SimulatedAnnealing	$1.6 \times 10^{1} \pm 7.2 \times 10^{-2}$	$9.3 \times 10^{0} \pm 0.0$
ParticleSwarm	$6.9 \times 10^{0} \pm 3.4 \times 10^{-2}$	$1.0 \times 10^{1} \pm 0.0$
DifferentialEvolution	$7.1 \times 10^{0} \pm 3.1 \times 10^{-2}$	$9.4 \times 10^{0} \pm 0.0$