Physical Design using Differentiable Learned Simulators

Kelsey R. Allen [*1]   Tatiana Lopez-Guevara [*1]   Kimberly Stachenfeld [*1]   Alvaro Sanchez-Gonzalez [1]
Peter Battaglia [1]   Jessica Hamrick [1]   Tobias Pfaff [1]

Abstract

Designing physical artifacts that serve a purpose—such as tools and other functional structures—is central to engineering as well as everyday human behavior. Though automating design has tremendous promise, general-purpose methods do not yet exist. Here we explore a simple, fast, and robust approach to inverse design which combines learned forward simulators based on graph neural networks with gradient-based design optimization. Our approach solves high-dimensional problems with complex physical dynamics, including designing surfaces and tools to manipulate fluid flows and optimizing the shape of an airfoil to minimize drag. This framework produces high-quality designs by propagating gradients through trajectories of hundreds of steps, even when using models that were pre-trained for single-step predictions on data substantially different from the design tasks. In our fluid manipulation tasks, the resulting designs outperformed those found by sampling-based optimization techniques. In airfoil design, they matched the quality of those obtained with a specialized solver. Our results suggest that despite some remaining challenges, machine learning-based simulators are maturing to the point where they can support general-purpose design optimization across a variety of domains.

1. Introduction

Humans are creators. Our ancestors created stone tools which led to innovations in hunting and food consumption, aqueducts and irrigation systems which revolutionized farming and urban habitation, and more recently, airplanes which let us cross the globe in hours. Automatically designing objects to exhibit a desired property—often referred to as *inverse design*—promises to transform science and engineering, including aerodynamics (Eppler, 2012), material design (Butler et al., 2016), optics (Colburn & Majumdar, 2021), and robotics (Gupta et al., 2021; Xu et al., 2021).

Despite its promise, widespread practice of inverse design has been limited by the availability of fast, general-purpose simulators. In science and engineering, many methods rely on specialized "classical" solvers, which are handcrafted to simulate a particular physical process. While accurate and reliable, these solvers can be quite slow, may not provide gradients, and are narrow in their applicability (Cranmer et al., 2020). In robotics and reinforcement learning, simulators are often learned, but accumulate errors over long time horizons and often struggle to generalize beyond their training data (Janner et al., 2019; Talvitie, 2014; Venkatraman et al., 2015), making them unsuitable for design optimization without further finetuning.

Recently, a class of learned physics simulators based on graph neural networks (GNNs) has been proposed (Pfaff et al., 2021; Sanchez-Gonzalez et al., 2020). These models have shown success in general-purpose physical prediction, exhibiting high accuracy and generalization ability. However, this may still be insufficient for inverse design problems, as optimizers can exploit regions in the state space where model predictions are unreliable (Lutter et al., 2021). Models must therefore be more than just accurate overall: they must also be robust and smooth. It is unknown whether GNN-based physics simulators exhibit these properties.

In this paper, we optimize physical designs by performing gradient descent through pretrained, GNN-based, state-of-the-art learned simulators. We use this approach to perform successful inverse design without requiring further finetuning of the simulator. Across two high-dimensional fluid manipulation tasks (2D FLUID TOOLS and 3D WATERCOURSE) and a design task from aerodynamics (AIRFOIL), we show that learned simulators: (1) produce high-quality designs across diverse physical tasks with complex particle- or mesh-based physics, while using the same underlying GNN architecture; (2) generalize sufficiently to permit designs far outside their training data; (3) support gradient-based optimization over hundreds of time steps, through states with thousands of particles, in tasks with up to 625
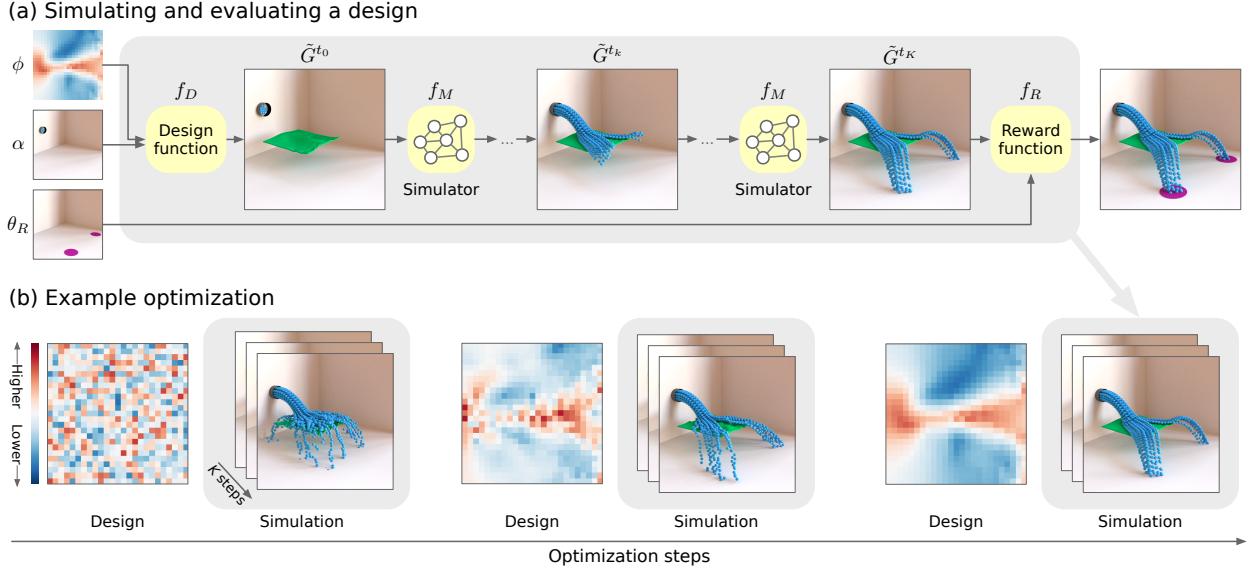
*Equal contribution. Authors listed alphabetically. [1]DeepMind, UK. Correspondence to: Kelsey R. Allen <krallen@deepmind.com>, Tatiana Lopez-Guevera <zepolitat@deepmind.com>, Kimberly Stachenfeld <stachenfeld@deepmind.com>, Tobias Pfaff <tpfaff@deepmind.com>.

(a) Simulating and evaluating a design

(b) Example optimization

Figure 1. Optimizing a physical design. Here, the goal is to direct a stream of water (shown in blue) into two "pools" (shown in purple) by designing a "landscape" (shown in green) parameterized as a 2D height field. (a) The simulation pipeline takes in a design ϕ and initial conditions α and uses the design function f_D to produce an initial state. The simulation is rolled out with a pre-trained learned simulator f_M for K steps, at which point the final state is passed (along with reward parameters θ_R) to the reward function f_R, which computes the quality of the design. (b) Each step of optimization involves rolling out the simulation and then adjusting the design (ϕ) accordingly using an optimizer such as gradient descent or CEM. Shown are selected frames from gradient-based optimization in the *2 Pools* task of the 3D WATERCOURSE domain.

design parameters (and as a result, produce better designs than sampling-based optimization using a classical simulator); and (4) can be much faster than specialized simulators used in engineering, while generating designs of similar quality. Overall, our results are a proof-of-concept for how state-of-the-art learned simulators can be used at scale to optimize designs for different physical tasks.

2. Background

Solving inverse problems with physical simulators has a long history in science and engineering, spanning data assimilation for weather modeling (Navon, 2009), system identification in robotics (Guevara et al., 2017; Seita et al., 2020), and tomographic and geophysical imaging (Cui et al., 2016; Pascual-Marqui, 1999). Inverse design can be framed as an inverse problem in which the objective is to optimize design parameters to produce some desired target property. Simulation-based inverse design has been studied in a variety of disciplines, including nanophotonics (Molesky et al., 2018), material science (Dijkstra & Luijten, 2021), mechanical design (Coros et al., 2013), and aerodynamics (Anderson & Venkatakrishnan, 1999; Rhie, 1983).

Classical numerical solvers used for inverse design can be highly accurate, but are often inefficient (making sampling-based inference methods infeasible for high-dimensional designs) and domain-specialized (prohibiting general-purpose

inverse design across domains (Choi et al., 2021)). Differentiable simulators (Freeman et al., 2021; Hu et al., 2019; Schenck & Fox, 2018) have recently garnered attention, as they allow for more sample-efficient gradient-based optimization. However, like classical solvers, they are still typically narrow in application scope, as many simulation techniques are hard to express as a differentiable program (e.g. constraint dynamics and multiphysics coupling).

The last few years have seen increased interest in using machine learning to accelerate inverse design across a variety of applications (e.g. Challapalli et al., 2021; Christensen et al., 2020; Gómez-Bombarelli et al., 2018; Forte et al., 2022; Hoyer et al., 2019; Kumar et al., 2020; Li et al., 2020a; Liu et al., 2018; Sha et al., 2021; Zheng et al., 2021). These methods can provide impressive speedups over classical approaches by using learned generative models to propose designs (thus calling an expensive simulator fewer times), or by learning a scoring function which maps designs to target values (replacing the simulator entirely). However, they are based on components with limited out-of-domain generalization, restricting new designs to configurations near the training data or requiring model refinement.

We instead propose to replace classical simulators with learned simulators. Like learned scoring functions, learned simulators can be faster than classical simulators (Kochkov et al., 2021; Stachenfeld et al., 2021) and are differentiable when parameterized as a neural network. Furthermore,

learned simulators mimic the underlying physical dynamics independent of the design task and are therefore more likely to generalize. Physics simulators have been successfully implemented as learned, differentiable models of complex dynamics such as fluids, rigid-body interactions, and soft-body systems (Bhatnagar et al., 2019; Li et al., 2020b; Mrowca et al., 2018; Rudy et al., 2017; Thuerey et al., 2020; Ummenhofer et al., 2020; Wang et al., 2020). Graph-based models in particular are promising candidates for design problems, having demonstrated high accuracy, stability, efficiency, and generalization performance (Belbute-Peres et al., 2020; Pfaff et al., 2021; Sanchez-Gonzalez et al., 2020).

However, high-quality forward models do not necessarily translate into better downstream task performance (Hamrick et al., 2020; Lutter et al., 2021). While learned simulators have been used successfully for planning and control in low-dimensional state spaces (up to 21 degrees of freedom (DOF), e.g. Bharadhwaj et al., 2020; Sanchez-Gonzalez et al., 2018; Wang et al., 2019) or more complex domains with small action spaces (6 DOF, e.g. Li et al., 2019), they require replanning at every timestep to avoid error accumulation. It is not known whether learned simulators can support gradient-based, high-dimensional inverse design which demands high accuracy, well-behaved gradients, long-term rollout stability, and generalization beyond the training data.

Here we study inverse design in non-rigid, graph-based physical systems with up to 625 design dimensions and 2000 state dimensions, over 50–300 timesteps, without replanning after the design period. We show that using gradient-based optimization with learned, general-purpose simulators is an effective choice for inverse design.

3. Problem Formulation

Consider the design task depicted in Figure 1, in which the goal is to direct a stream of water (shown in blue) into two "pools" (shown in purple) by designing a "landscape" (shown in green) parameterized as a 2D height field. Here, an ideal design will create ridges and valleys that direct fluid into the two targets. In the next sections, we formalize what it means to find and evaluate such a design and discuss our choices for simulator and optimizer.

3.1. Learned simulators

To demonstrate the utility of learned simulators for finding physical designs, we rely on the recently developed MESHGRAPHNETS model (Pfaff et al., 2021), which is an extension of the GNS model for particle simulation (Sanchez-Gonzalez et al., 2020). MESHGRAPHNETS is a type of message-passing graph neural network (GNN) that performs both edge and node updates (Battaglia et al., 2018; Gilmer et al., 2017), and which was designed specifically

for physics simulation. Here, we briefly summarize how the learned simulator works, and refer interested readers to the original papers for details.

We consider simulations over physical states represented as graphs $G \in \mathcal{G}$. The state $G = (V, E)$ has nodes V connected by edges E, where each node $v \in V$ is associated with a position $\mathbf{u}_v$ and additional dynamical quantities $\mathbf{q}_v$. These graphs may be either meshes (as in MESHGRAPH-NETS) or particle systems (as in GNS). In a mesh-based system (such as AIRFOIL), V and E correspond to vertices and edges in the mesh, respectively. In a particle system (such as 2D FLUID TOOLS), each node corresponds to a particle and edges are computed dynamically based on proximity. Under this framework, we can also consider hybrid mesh-particle systems (such as 3D WATERCOURSE). See Appendix B for model implementation details, and Appendix C for details on the representation used for each domain.

The simulation dynamics are given by a "ground-truth" simulator $f_S : \mathcal{G} \to \mathcal{G}$ which maps the state at time t to that at time $t + \Delta t$. The simulator f_S can be applied iteratively over K time steps to yield a trajectory of states, or a "rollout," which we denote $(G^{t_0}, ..., G^{t_K})$. Using MESH-GRAPHNETS, we learn an approximation f_M of the ground-truth simulator f_S. The learned simulator f_M can be similarly applied to produce rollouts $(\tilde{G}^{t_0}, \tilde{G}^{t_1}, ..., \tilde{G}^{t_K})$, where $\tilde{G}^{t_0} = G^{t_0}$ represents initial conditions given as input. We note that a learned simulator allows us to take much larger time-steps than f_S, permitting shorter rollout lengths: in our running example, one model step corresponds to 200 internal steps of the classical simulator. See Figure 1a for an illustration of simulation using a learned model.

3.2. Optimizing design parameters

To optimize a physical design, we leverage the pipeline shown in Figure 1: (1) transform design parameters into an initial scene, (2) simulate the scene using f_M or f_S, (3) evaluate how well the simulation achieves the desired behavior, and (4) adjust the design parameters accordingly.

Design parameters To produce the initial state G^{t_0}, we introduce a differentiable design function $f_D : \Phi \times \mathcal{A} \to \mathcal{G}$ which maps design parameters $\phi \in \Phi$ and other initial conditions $\alpha \in \mathcal{A}$ to an initial state: $G^{t_0} = f_D(\phi, \alpha)$. In our landscape design task (Figure 1), ϕ is the 2D height field of the mesh, while α is the non-controllable objects in the scene like the initial position of the fluid.

Maximizing reward The reward function $f_R : \mathcal{G} \times \Theta_R \to \mathbb{R}$ maps the final state of a length-K trajectory (G^{t_K} or $\tilde{G}^{t_K}$) and parameters $\theta_R \in \Theta_R$ to a scalar value. In our running example, the reward function is defined as the Gaussian likelihood of each fluid particle under the closest "pool", averaged across particles (Figure 1a).
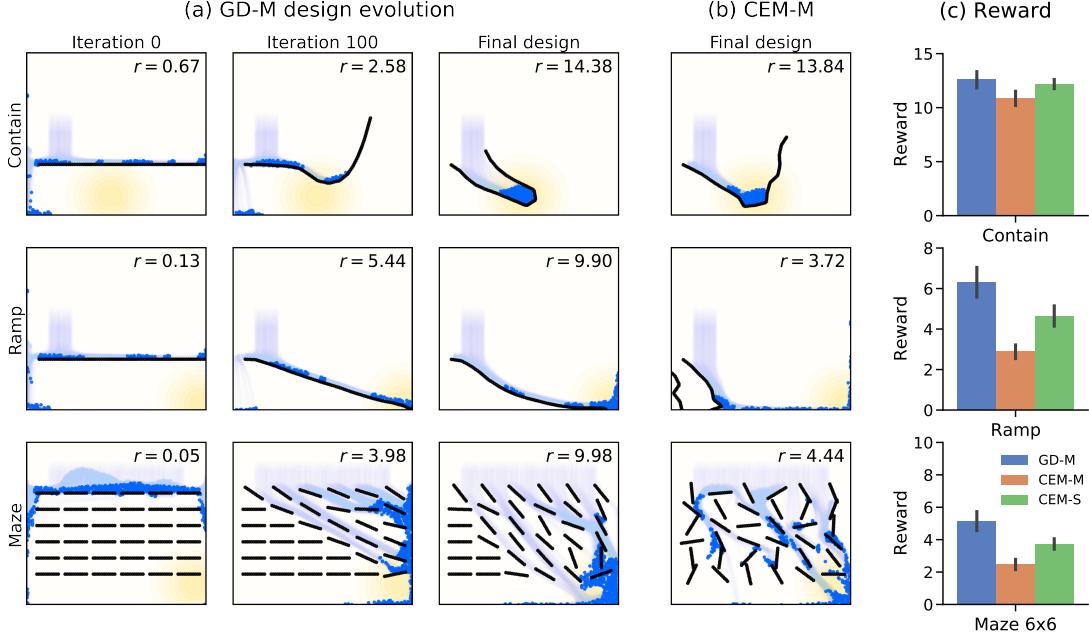
Figure 2. 2D FLUID TOOLS results. The state spaces consist of 10^2–10^3 particles and the design spaces of 16–36 parameters. (a) Evolution of designs found by GD-M during optimization for each 2D FLUID TOOLS task. Visualizations correspond to simulations of the designs under f_S. The design is shown in black, fluid particles in blue, and Gaussian reward in yellow. The transparent particles show the location of fluid for $t < t_K$, and the solid particles show the location of fluid at the final frame ($K = 150$). r denotes reward for the current design. (b) Final designs found by CEM-M. (c) Mean reward over 50 reward locations (with bootstrapped 95% confidence intervals) obtained by each optimizer across the 2D FLUID TOOLS tasks. For *Contain* and *Ramp*, results are shown for 16 joints; for *Maze*, results are shown for a 6×6 grid of 36 rotors. Across these tasks, GD-M outperforms both CEM-M and CEM-S.

We define the full objective under the ground-truth simulator f_S as $J_S(\phi) := f_R(f_S^{(K)}(f_D(\phi, \alpha)); \theta_R)$, where $f_S^{(K)}$ indicates K applications of the simulator. We want to find the design parameters that maximize J_S, i.e. $\phi^* = \text{argmax}_\phi J_S(\phi)$. We can approximate this optimization using a learned simulation model instead by maximizing $J_M(\phi) := f_R(f_M^{(K)}(f_D(\phi, \alpha)); \theta_R)$.

Optimizers Optimal design parameters ϕ^* can be found using any generic optimization technique. Given the differentiability of the learned simulator f_M, we are particularly interested in evaluating gradient-based optimization, which requires fewer function evaluations and scales better to large design spaces than sampling-based techniques (Bharadhwaj et al., 2020). We focus on the Adam optimizer (Kingma & Ba, 2015), which we use to find ϕ^* by computing the gradient $\nabla_\phi J_M(\phi)$. This involves backpropagating gradients through the reward function f_R, length-K rollout produced by $f_M^{(K)}$, and design function f_D.

As a baseline, we consider the cross-entropy method (CEM) (Rubinstein & Kroese, 2004), a gradient-free sampling-based technique that is popular in model-based control (Chua et al., 2018; Wang et al., 2019). CEM can be used with any simulator and works by sampling a population of

candidates for ϕ and evolving them to maximize the reward. However, CEM requires multiple evaluations of f_M or f_S per optimizer step (depending on the population size, which is 20–40), whereas Adam considers only a single candidate ϕ and thus only a single evaluation per step.

Across our design tasks (Section 4) we compared: gradient descent with the learned simulator (**GD-M**), CEM with the learned simulator (**CEM-M**), and CEM with the ground-truth simulator (**CEM-S**). In all tasks, f_S is non-differentiable, preventing a comparison to GD-S. However, in the special case of AIRFOIL, we compare to **DAFoam** (He et al., 2020), a specialized solver which computes gradients with the adjoint method.

Evaluation Unless otherwise noted, we always evaluate the quality of an optimized design ϕ^* using the ground-truth objective $J_S(\phi^*)$, regardless of whether ϕ^* was found using the learned model f_M (as in CEM-M and GD-M) or the ground-truth simulator f_S (as in CEM-S). We also use rollouts from f_S to produce visualizations in the figures.

4. Design Tasks

We formulated a set of design tasks across three different physical domains with high-dimensional state spaces and
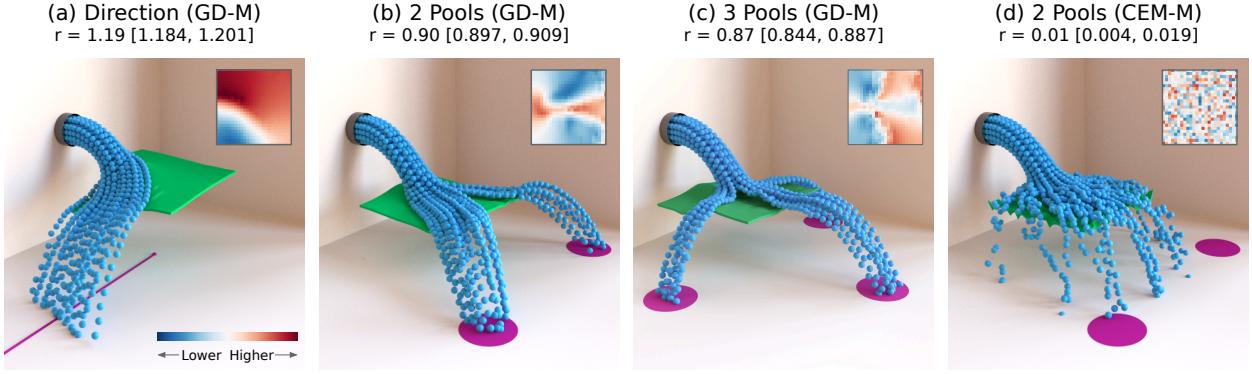
Figure 3. 3D WATERCOURSE results found with GD-M or CEM-M and evaluated with f_S. Simulations use up to 2000 particles and 625 design parameters. The heightmap of a 2D landscape is optimized to redirect the fluid towards the purple targets; birds eye views of heightmaps are shown in the upper right corner of each subplot. In this high-dimensional task domain, GD finds designs with high reward (a-c), while CEM fails to find meaningful solutions for *2 Pools* (d) as well as the other tasks (Figure A.5a). Each subplot reports the mean reward (r) and bootstrapped [lower, upper] 95% confidence intervals for the corresponding optimizer and task (averaged over 10 randomized initial designs for each task variation, see Section C.2).

complex dynamics. Each domain uses a different ground-truth simulator f_S, which is used to evaluate designs and pre-train the learned simulator model f_M.

4.1. 2D FLUID TOOLS

Inspired by existing 2D physical reasoning benchmarks (Allen et al., 2020; Bakhtin et al., 2019), these tasks involve creating one or more 2D "tool" shapes to direct fluid into a particular goal region (Figure 2, Section C.1). We consider three tasks with 4–48 design parameters where the goal is to guide the fluid such that each particle comes as close as possible to the center of a randomly-sampled yellow reward region. In *Contain*, the joint angles ϕ_{joints} of a multi-segment tool must be optimized to catch the fluid by creating cup- or spoon-like shapes. In *Ramp*, the joint angles ϕ_{joints} of a multi-segment tool must be optimized to guide the fluid to a distant location. In *Maze*, the rotation ϕ_{rot} of multiple tools must be optimized to funnel the fluid to the target location. Fluid dynamics are represented using 10^2–10^3 particles, and unrolled for up to 300 time steps.

The learned model f_M is trained on the 2D WaterRamps dataset released by Sanchez-Gonzalez et al. (2020) which uses the solver in Hu et al. (2018) to generate trajectories. The dataset contains scenes with 1–4 straight line segments; no curved lines or large numbers of obstacles are shown. Therefore, designs which solve 2D FLUID TOOLS tasks are necessarily far out of distribution for the learned model.

4.2. 3D WATERCOURSE

To evaluate higher-dimensional inverse design, we created a "landscaping" task that requires optimizing a 3D surface to guide fluids into different areas of an environment (Figure 3, Section C.2). Specifically, water flows out of a pipe and onto

an obstacle parameterized by ϕ_{map}, a 25×25 heightmap (625 design parameters). In *Direction*, ϕ_{map} is optimized to redirect the fluid stream towards a specified direction. In *2 Pools* and *3 Pools*, ϕ_{map} is optimized to split the fluid stream such that particles hitting the floor land as close as possible to one of two or three specified pools. Each task has multiple variants, such as different target directions in *Direction*. The simulation is unrolled for 50 time steps, and contains up to 2048 particles. The learned model f_M is trained on data generated from the simulator in Bender & Koschier (2015).

4.3. AIRFOIL

Shape optimization in aerodynamics is one area where gradient-based optimization is routinely applied using traditional simulators (Buckley et al., 2010). Here, we consider the well-studied task of drag optimization of a 2D airfoil profile (Figure 4, Section C.3). In this task, a wing is defined using a curve on a 2D mesh, which can be deformed using a set of 10 control points ϕ_{ctrl}. The reward function is formulated as optimizing the wing shape to minimize drag under certain constraints such as constant lift and bounds on the shape to prevent degenerate (e.g. infinitely thin) configurations. Lift and drag coefficients are computed by running an aerodynamics simulation on a 4158 node mesh. The learned model f_M is trained on data generated from the ground-truth simulator f_S, for which we use the OpenFOAM solver (OpenCFD Ltd, 2021).

4.4. Model learning and optimization

While each domain has a different state space structure and uses a different ground-truth simulator f_S, the learned simulators f_M all share the same architecture (with identical
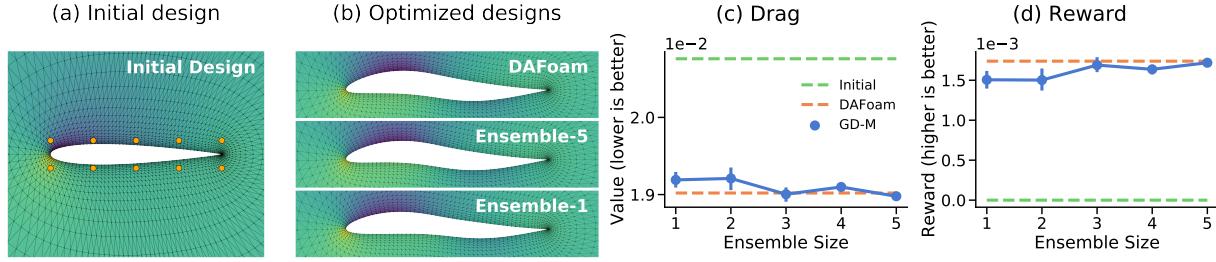
Figure 4. AIRFOIL results. (a) An initial airfoil design is warped by moving 10 control points (orange dots). The physics model simulates the resulting aerodynamics on a 4158 node mesh, based on which lift and drag are computed. (b) For the task of finding a minimum-drag configuration under constant lift constraint, gradient-based learned design is able to find similar designs to specialized solver DAFoam, both using single models and ensembles. (c-d) Larger ensemble sizes of 3–5 achieve a close quantitative match to DAFoam for both drag and overall reward. Shown are means over 10 randomized initial designs, with bootstrapped 95% confidence intervals.

hyperparameters in 2D FLUID TOOLS and 3D WATER-COURSE, and only minor variations of the hyperparameters for AIRFOIL due to it being a steady state simulation; see Appendix B). The models are trained for next-step prediction on task-independent datasets (random perturbations of the design space for AIRFOIL and 3D WATERCOURSE, and an open-source, qualitatively distinct dataset for 2D FLUID TOOLS), and are unrolled for up to 300 time steps during design optimization without further fine-tuning (see Section 3.2 for details).

5. Results

Our results show that learned simulators can be used to effectively optimize various designs despite significant domain shift and long rollout lengths. The same underlying model architecture is used for each domain, highlighting the generality of learned simulators for design. Examples of designs found with our approach are available at: `https://sites.google.com/view/optimizing-designs`. Performance is always evaluated using the ground-truth simulator f_S (see Section 3.2). Here we discuss these results, and compare the capabilities of gradient descent with learned simulators over classical simulators and sampling-based optimization techniques.

5.1. Overall results

We first asked whether a learned simulator combined with gradient descent (GD-M) could produce good-quality designs at all. This approach might fail in various ways: accumulating model error, vanishing or exploding gradients (Bengio et al., 1994), or domain shift (Hamrick et al., 2020). However, as the following results show, GD-M produced high-quality designs across all three domains.

Figure 2a shows qualitative results for 2D FLUID TOOLS (Section 4.1), where our approach (GD-M) produces intuitive, functional designs to contain (*Contain*), transport (*Ramp*), or funnel (*Maze*) the fluid to a target location. On

average, GD-M outperforms CEM-M by 16.1–118.9%, indicating a substantial benefit of gradient-based optimization. GD-M also outperforms CEM-S by 3.9–37.5%, despite using a learned simulator rather than the ground-truth. However, these design spaces are still relatively small (between 16 and 36 dimensions). In 3D WATERCOURSE (Section 4.2), we substantially increase the dimensionality to a 625-dimensional landscape. Here, GD-M produces robust designs, creating ridges to re-route water in particular directions or valleys to direct water into pools (Figure 3a-c). In comparison, CEM-M cannot solve any of these tasks, with performance 30–85× worse than GD-M.

In AIRFOIL (Figure 4b), GD-M recovers the characteristic S-curve shape for a low-drag airfoil under a small angle of attack, and matches the design obtained with DAFoam, an adjoint aerodynamics solver which computes close-to-optimal designs for this task. Specifically, the design obtained with DAFoam yields a drag coefficient of 0.01902, while GD-M finds designs with drag between 0.01898–0.01919 depending on ensemble size (see Section 5.4). Importantly, DAFoam's solver and optimizer are highly specialized for the particular task of airfoil design, while our approach is more general-purpose in that it requires only trajectory data for training and a generic gradient-based optimizer.

5.2. Model stability & gradient quality

We investigated accuracy over long timescales by measuring the effect of rollout length on design quality in 2D FLUID TOOLS (Figure 5a and A.10). Longer rollouts can in principle allow for higher reward in this task as they give the fluid time to settle; however, with learned models, they can also be unstable due to error accumulation (Talvitie, 2014; Venkatraman et al., 2015). Nevertheless, we find that the learned simulator does not seem to be severely impacted by this problem. Specifically, the quality of designs found by GD-M increases up to 225 steps (Figure 5a), indicating that the learned simulator's accuracy and gradients remain stable for a surprisingly long time. Across the episode lengths eval-
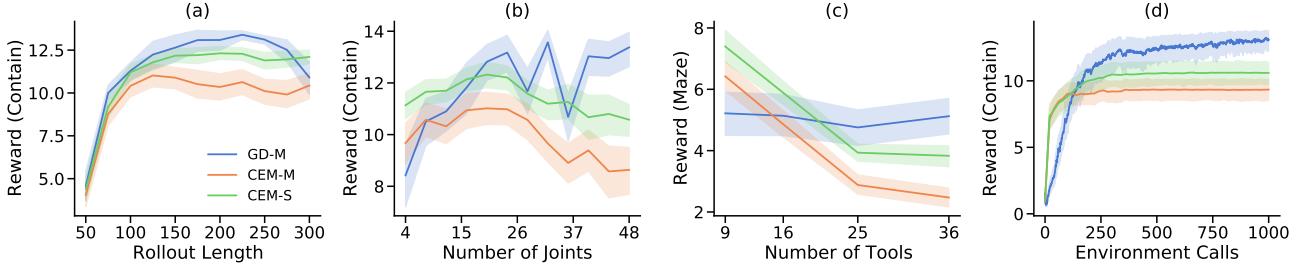
Figure 5. Ablation experiments on the *Contain* and *Maze* tasks. (a) Performance of all optimizers increase with rollout length; GD-M performance starts to deteriorate around step 225. (b) In *Contain*, CEM performance drops when increasing the number of joints above 24, while GD-M remains stable. (c) We observe a similar trend with the number of tools in *Maze*. (d) CEM often gets stuck in sub-optimal solutions early in optimization, while GD-M performance continues to increase.

uated, we find that GD-M outperforms not only CEM-M (by 18.1% on average) but also CEM-S (by 4.4% on average). This indicates that the benefits of a having a learned model that supports better optimization techniques can outweigh the error incurred by long rollouts.

The strong performance of GD-M on longer rollout lengths is noteworthy. Gradients tend to degrade when passed through chains of many model evaluations, and as a result, previous work generally only optimizes gradients in small action spaces over just a few time-steps (Li et al., 2019). We speculate that one reason for the success of GD-M is the addition of noise in training f_M, which promotes stability on the forward pass and may also force smoother gradients.

5.3. Generalization

Deep networks often struggle to generalize far from their training data (Geirhos et al., 2018). This poses a problem for design: to produce in-distribution training data, we would already need to know what good designs look like, thus defeating the aim of wanting to find *new* designs. However, we find that the GNN-based simulators studied here overcome this issue. As noted in Section 4.1, the learned simulator for 2D FLUID TOOLS was trained on a pre-existing, highly simplified dataset where only one to four straight line segments interact with a fluid (Appendix C). In contrast, the design tasks studied here involve highly articulated, curved obstacles (*Contain*, *Ramp*) or a larger number of obstacles (*Maze*); yet, GD-M still discovers effective designs without requiring any finetuning (Figure 2). We suspect this is because the model is trained to learn local collision rules, making it more robust to global distribution shift.

Using a learned simulator trained in a relatively simple environment has another unexpected advantage. In rare cases, classical simulators suffer from degenerate behavior around certain edge cases. For example, with the classical simulator for 2D FLUID TOOLS, particles can get stuck in between joint segments, especially when there are a large number of parts or joints (Figure A.9). However, since the learned

simulator was trained on simpler data where these effects are unobserved, it picks up only on the appropriate collision performance and not the unrealistic edge cases. Thus, the learned simulator produces more plausible rollouts than the classical simulator in these cases, and might therefore be a better candidate for producing designs that would transfer to the real world.

5.4. Improving accuracy with ensembles

In engineering tasks like AIRFOIL, simulators must be especially accurate, as small differences in the predicted pressure field can cause large errors in lift and drag coefficients. While GD-M (without ensembling) can produce designs close to DAFoam's, we notice a slightly rounder wing front (Figure 4b, bottom) causing a small increase in drag (0.01919 versus 0.01902 in DAFoam).

To further improve performance, we implemented an ensemble of learned simulators trained on separate splits of the training set. Ensembles are a popular choice for training transition models for use in control (Chua et al., 2018), as they can provide higher quality predictions and are more resistant to delusions—a particularly problematic issue for accuracy-sensitive domains such as airfoil design. During optimization, we make predictions with all models in the ensemble, each trained on a different data split, and average the gradients. As shown in Figure 4b-c, larger ensembles yield designs with significantly lower drag ($\beta = -5.3 \times 10^{-5}$, $p = 0.0003$, where β is a linear regression coefficient) and higher overall reward ($\beta = 5.6 \times 10^{-5}$, $p = 0.0001$), and are able to produce designs very close to the solution found by DAFoam, with a drag coefficient of 0.01898 (size-5 ensemble). Thus, with ensembles, we are able to achieve performant designs with a general-purpose learned simulator, indicating that we can use learned models for design optimization in spaces traditionally reserved for specialized solvers like DAFoam.

5.5. Scalability to larger design spaces

In larger design spaces, sampling-based optimization procedures quickly become intractable, especially with relatively slow simulators. We hypothesized that gradient descent with fast, learned simulators could overcome this issue, especially as the size of the design space is increased. We therefore compared different optimizers on 2D FLUID TOOLS as a function of the dimensionality of the design space (the number of tool joints in *Contain* or the number of tools in *Maze*) and on the higher-dimensional 3D WATERCOURSE.

For *Contain* (Figure 5b), the performance of GD-M increases with the number of joints as increasingly fine grained solutions are made possible. In contrast, for both CEM-M and CEM-S, design quality deteriorates with more joints as high quality solutions become harder to find with random sampling. In the highest dimensional *Contain* task with 48 tool joints, GD-M outperforms CEM-M by 154.9% and CEM-S by 126.5%. When CEM does find solutions (Figure 2b), they lack global coherence and appear more jagged than solutions found with GD-M. Similarly, for *Maze* (Figure 5c), the performance of GD-M is largely unaffected by the number of tools, while the performance of CEM-M and CEM-S both degrade as the design space grows. For the highest dimensional *Maze* problem with 36 joints, GD-M outperforms CEM-M by 207.4% and CEM-S by 133.7%.

In the 625-dimensional 3D WATERCOURSE task, CEM-M performs $30–85\times$ worse than GD-M (Figure 3) despite extensive hyperparameter tuning. This trend held across all tasks (Figure A.5a), and even persisted when using fewer control points in the design space (Figure A.6). This is due not only to 3D WATERCOURSE's larger design space, but also because this problem requires a globally coherent solution: modifying small areas independently is unlikely to have much effect on the global movement of the fluid.

5.6. Model speed and sample efficiency

Learned simulators can provide large speedups over traditional simulators in certain domains by learning to compensate for coarser sub-stepping and making optimal use of hardware acceleration. In AIRFOIL, although we use a very simple GD setup, our approach is able to find very similar designs as DAFoam's specialized optimizer. Moreover, our approach requires only 21s (single model) to 62s (size-5 ensemble) on a single A100 GPU, compared to 1021s for DAFoam run on an 8-core workstation, despite requiring $10\times$ more optimization steps.

In 2D FLUID TOOLS, the ground-truth simulator runs at a similar speed to the learned model (see Sanchez-Gonzalez et al., 2020), but is non-differentiable and therefore depends on more expensive gradient-free optimization techniques which require more function evaluations. We use 20–40 function evaluations per optimization step of CEM, compared to a single evaluation with GD (which is about $3\times$ more costly, due to the gradient computation). Thus, GD with a differentiable learned model can be much more efficient than using the ground truth simulator with a sampling-based method.

6. Discussion

We used state-of-the-art learned, differentiable physics simulators with gradient-based optimization to solve challenging inverse design problems. Across three domains and seven tasks, which involved designing landscapes and tools to control water flows or optimizing the shape of an airfoil, we demonstrated that gradient descent with pre-trained simulators can discover high-quality designs that match or exceed the quality of those found using alternative methods. This approach succeeds in a variety of interesting and surprising ways: it permits gradient backpropagation through complex physical trajectories for hundreds of steps; scales to tasks with large design and state spaces (100s and 1000s of dimensions, respectively); and successfully generates designs which require the learned simulator to generalize far beyond its training data. In the classic aerodynamics problem of airfoil shape optimization, our approach produces a design comparable to that of a specialized solver using only simple, general-purpose strategies like model ensembling.

While our results have exciting implications for inverse design, they also open up possibilities for explaining everyday human behavior like tool invention—a longstanding puzzle in cognitive science (Allen et al., 2020; Osiurak & Badets, 2016; Shumaker et al., 2011). With general-purpose learned simulators, we have the potential not just to create highly specialized tools in engineering domains, but also to model everyday tool creation, such as creating a hook from a pipe cleaner, building a blanket fort, or folding a paper boat.

Our approach has limitations that should be visited in future work. Gradient descent is inappropriate for design spaces with regions of zero gradients, such as in 2D FLUID TOOLS tasks where the fluid may not always make contact with the tool (Figure A.8). Many interesting design tasks also have variably-sized or combinatorial design spaces that cannot easily be optimized with gradient descent, such as computer-aided design (CAD) approaches to 3D modeling. An exciting future direction will be to integrate general-purpose learned simulators with hybrid optimization techniques such as those used in material science and robotics (Chen & Gu, 2020; Toussaint et al., 2018). As learned simulators continue to improve, we could also use them to do even broader cross-domain, multi-physics design. While challenges remain, our results represent a promising step towards faster and more general-purpose inverse design.

References

Allen, K. R., Smith, K. A., and Tenenbaum, J. B. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 2020. Cited on pages 5 and 8.

Anderson, W. K. and Venkatakrishnan, V. Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation. *Computers & Fluids*, 28(4-5): 443–480, 1999. Cited on page 2.

Bakhtin, A., van der Maaten, L., Johnson, J., Gustafson, L., and Girshick, R. Phyre: A new benchmark for physical reasoning. *Advances in Neural Information Processing Systems*, 32:5082–5093, 2019. Cited on page 5.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. Cited on page 3.

Belbute-Peres, F. d. A., Economon, T. D., and Kolter, J. Z. Combining differentiable PDE solvers and graph neural networks for fluid flow prediction. In *Proceedings of the 37th International Conference on Machine Learning ICML 2020*, 2020. Cited on page 3.

Bender, J. and Koschier, D. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics symposium on computer animation*, pp. 147–155, 2015. Cited on pages 5 and 17.

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. Cited on page 6.

Bharadhwaj, H., Xie, K., and Shkurti, F. Model-predictive control via cross-entropy and gradient-based optimization. In *Learning for Dynamics and Control*, pp. 277–286. PMLR, 2020. Cited on pages 3 and 4.

Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K., and Kaushik, S. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019. Cited on page 3.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. Cited on page 14.

Buckley, H. P., Zhou, B. Y., and Zingg, D. W. Airfoil optimization using practical aerodynamic design requirements. *Journal of Aircraft*, 47(5):1707–1719, 2010. Cited on page 5.

Butler, K. T., Frost, J. M., Skelton, J. M., Svane, K. L., and Walsh, A. Computational materials design of crystalline solids. *Chemical Society Reviews*, 45(22):6138–6146, 2016. Cited on page 1.

Challapalli, A., Patel, D., and Li, G. Inverse machine learning framework for optimizing lightweight metamaterials. *Materials & Design*, 208:109937, 2021. Cited on page 2.

Chen, C.-T. and Gu, G. X. Generative deep neural networks for inverse materials design using backpropagation and active learning. *Advanced Science*, 7(5):1902607, 2020. Cited on page 8.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. Cited on page 14.

Choi, H., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl, F., Hodgins, J., Jain, A., Leve, F., et al. On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, 118(1), 2021. Cited on page 2.

Christensen, T., Loh, C., Picek, S., Jakobović, D., Jing, L., Fisher, S., Ceperic, V., Joannopoulos, J. D., and Soljačić, M. Predictive and generative machine learning models for photonic crystals. *Nanophotonics*, 9(13):4183–4192, 2020. Cited on page 2.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018. Cited on pages 4 and 7.

Colburn, S. and Majumdar, A. Inverse design and flexible parameterization of meta-optics using algorithmic differentiation. *Communications Physics*, 4(1):1–11, 2021. Cited on page 1.

Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R. W., Matusik, W., and Bickel, B. Computational design of mechanical characters. *ACM Trans. Graph.*, 32(4), jul 2013. Cited on page 2.

Cranmer, K., Brehmer, J., and Louppe, G. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020. Cited on page 1.

Cui, Z., Wang, Q., Xue, Q., Fan, W., Zhang, L., Cao, Z., Sun, B., Wang, H., and Yang, W. A review on image reconstruction algorithms for electrical capacitance/resistance tomography. *Sensor Review*, 2016. Cited on page 2.

Dijkstra, M. and Luijten, E. From predictive modelling to machine learning and reverse engineering of colloidal

self-assembly. *Nature Materials*, 20(6):762–773, 2021. Cited on page 2.

Eppler, R. *Airfoil design and data*. Springer Science & Business Media, 2012. Cited on page 1.

Forte, A. E., Hanakata, P. Z., Jin, L., Zari, E., Zareei, A., Fernandes, M. C., Sumner, L., Alvarez, J., and Bertoldi, K. Inverse design of inflatable soft membranes through machine learning. *Advanced Functional Materials*, pp. 2111610, 2022. Cited on page 2.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. Brax-a differentiable physics engine for large scale rigid body simulation. 2021. Cited on page 2.

Geirhos, R., Temme, C. R. M., Rauber, J., Schütt, H. H., Bethge, M., and Wichmann, F. A. Generalisation in humans and deep neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 7549–7561, 2018. Cited on page 7.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017. Cited on page 3.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 02 2018. Cited on page 2.

Guevara, T. L., Taylor, N. K., Gutmann, M. U., Ramamoorthy, S., and Subr, K. Adaptable pouring: Teaching robots not to spill using fast but approximate fluid simulation. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2017. Cited on page 2.

Gupta, A., Savarese, S., Ganguli, S., and Fei-Fei, L. Embodied intelligence via learning and evolution. *Nature Communications*, 12(1):5721, Oct 2021. Cited on page 1.

Hamrick, J. B., Friesen, A. L., Behbahani, F., Guez, A., Viola, F., Witherspoon, S., Anthony, T., Buesing, L., Veličković, P., and Weber, T. On the role of planning in model-based deep reinforcement learning. *arXiv preprint arXiv:2011.04021*, 2020. Cited on pages 3 and 6.

He, P., Mader, C. A., Martins, J. R., and Maki, K. J. Dafoam: An open-source adjoint framework for multidisciplinary design optimization with openfoam. *AIAA journal*, 58(3): 1304–1319, 2020. Cited on page 4.

Hessel, M., Budden, D., Viola, F., Rosca, M., Sezener, E., and Hennigan, T. Optax: composable gradient transformation and optimisation, in JAX!, 2020. Cited on page 13.

Hoyer, S., Sohl-Dickstein, J., and Greydanus, S. Neural reparameterization improves structural optimization. *CoRR*, abs/1909.04240, 2019. Cited on page 2.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., and Jiang, C. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4): 1–14, 2018. Cited on pages 5, 14, and 21.

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., and Durand, F. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019. Cited on page 2.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019. Cited on page 1.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. Cited on page 4.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021. Cited on page 2.

Kumar, S., Tan, S., Zheng, L., and Kochmann, D. M. Inverse-designed spinodoid metamaterials. *Nature Computational Materials*, 6(1):73, 2020. Cited on page 2.

Ladson, C. L. *Effects of independent variation of Mach and Reynolds numbers on the low-speed aerodynamic characteristics of the NACA 0012 airfoil section*, volume 4074. National Aeronautics and Space Administration, Scientific and Technical . . . , 1988. Cited on page 17.

Li, J., Zhang, M., Martins, J., and Shu, C. Efficient aerodynamic shape optimization with deep-learning-based geometric filtering. *AIAA Journal*, 58:1–17, 07 2020a. Cited on page 2.

Li, Y., Wu, J., Tedrake, R., Tenenbaum, J. B., and Torralba, A. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2019. Cited on pages 3 and 7.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. M., and Anandkumar, A.

Fourier neural operator for parametric partial differential equations. *CoRR*, abs/2010.08895, 2020b. URL https://arxiv.org/abs/2010.08895. Cited on page 3.

Liu, Z., Zhu, D., Rodrigues, S. P., Lee, K.-T., and Cai, W. Generative model for the inverse design of metasurfaces. *Nano Letters*, 18(10):6570–6576, Sep 2018. ISSN 1530-6992. Cited on page 2.

Lutter, M., Hasenclever, L., Byravan, A., Dulac-Arnold, G., Trochim, P., Heess, N., Merel, J., and Tassa, Y. Learning dynamics models for model predictive agents. *arXiv preprint arXiv:2109.14311*, 2021. Cited on pages 1 and 3.

Molesky, S., Lin, Z., Piggott, A. Y., Jin, W., Vucković, J., and Rodriguez, A. W. Inverse design in nanophotonics. *Nature Photonics*, 12(11):659–670, 2018. Cited on page 2.

Mrowca, D., Zhuang, C., Wang, E., Haber, N., Fei-Fei, L., Tenenbaum, J. B., and Yamins, D. L. K. Flexible neural representation for physics prediction. *CoRR*, abs/1806.08047, 2018. Cited on page 3.

Navon, I. M. Data assimilation for numerical weather prediction: a review. *Data assimilation for atmospheric, oceanic and hydrologic applications*, pp. 21–65, 2009. Cited on page 2.

OpenCFD Ltd. OpenFOAM CFD solver. www.openfoam.org, 2021. Cited on pages 5 and 17.

Osiurak, F. and Badets, A. Tool use and affordance: Manipulation-based versus reasoning-based approaches. *Psychological review*, 123(5):534, 2016. Cited on page 8.

Pascual-Marqui, R. D. Review of methods for solving the eeg inverse problem. *International journal of bioelectromagnetism*, 1(1):75–86, 1999. Cited on page 2.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. Cited on pages 1, 3, and 14.

Reid, J. Free-form deformation. dafoam.github.io/docs/FFD/main.pdf, 2021. Cited on page 17.

Rhie, C Chow, L. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21 (11):1525–1532, 1983. Cited on page 2.

Rubinstein, R. Y. and Kroese, D. P. The cross-entropy method: A unified approach to monte carlo simulation, randomized optimization and machine learning. *Information Science & Statistics, Springer Verlag, NY*, 2004. Cited on page 4.

Rudy, S. H., Brunton, S. L., Proctor, J. L., and Kutz, J. N. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017. doi: 10.1126/sciadv.1602614. URL https://www.science.org/doi/abs/10.1126/sciadv.1602614. Cited on page 3.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479. PMLR, 2018. Cited on page 3.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020. Cited on pages 1, 3, 5, 8, 14, and 16.

Schenck, C. and Fox, D. Spnets: Differentiable fluid dynamics for deep neural networks. *Conference on Robot Learning (CoRL)*, 2018. Cited on page 2.

Secco, N., Kenway, G. K. W., He, P., Mader, C. A., and Martins, J. R. R. A. Efficient mesh generation and deformation for aerodynamic shape optimization. *AIAA Journal*, 2021. Cited on page 17.

Seita, D., Ganapathi, A., Hoque, R., Hwang, M., Cen, E., Tanwani, A. K., Balakrishna, A., Thananjeyan, B., Ichnowski, J., Jamali, N., Yamane, K., Iba, S., Canny, J., and Goldberg, K. Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. Cited on page 2.

Sha, W., Li, Y., Tang, S., Tian, J., Zhao, Y., Guo, Y., Zhang, W., Zhang, X., Lu, S., Cao, Y.-C., and Cheng, S. Machine Learning in polymer informatics. *InfoMat*, 3(4):353–361, 2021. Cited on page 2.

Shumaker, R. W., Walkup, K. R., and Beck, B. B. *Animal tool behavior: the use and manufacture of tools by animals*. JHU Press, 2011. Cited on page 8.

Stachenfeld, K., Fielding, D. B., Kochkov, D., Cranmer, M., Pfaff, T., Godwin, J., Cui, C., Ho, S., Battaglia, P., and Sanchez-Gonzalez, A. Learned coarse models for efficient turbulence simulation. *arXiv preprint arXiv:2112.15275*, 2021. Cited on page 2.

Talvitie, E. Model regularization for stable sample rollouts. In *UAI*, pp. 780–789, 2014. Cited on pages 1 and 6.

Thuerey, N., Weißenow, K., Prantl, L., and Hu, X. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020. Cited on page 3.

Toussaint, M. A., Allen, K. R., Smith, K. A., and Tenen-
baum, J. B. Differentiable physics and stable modes for
tool-use and manipulation planning. In *Proceedings of the
Robotics: Science and Systems, RSS 2018*, 2018. Cited
on pages 8 and 20.

Ummenhofer, B., Prantl, L., Thürey, N., and Koltun, V.
Lagrangian fluid simulation with continuous convolutions.
In *International Conference on Learning Representations*,
2020. Cited on page 3.

Venkatraman, A., Hebert, M., and Bagnell, J. A. Improving
multi-step prediction of learned time series models. In
Twenty-Ninth AAAI Conference on Artificial Intelligence,
2015. Cited on pages 1 and 6.

Wang, R., Kashinath, K., Mustafa, M., Albert, A., and Yu,
R. Towards physics-informed deep learning for turbulent
flow prediction, 2020. Cited on page 3.

Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Lan-
glois, E., Zhang, S., Zhang, G., Abbeel, P., and Ba,
J. Benchmarking model-based reinforcement learning.
arXiv preprint arXiv:1907.02057, 2019. Cited on pages 3
and 4.

Xu, J., Chen, T., Zlokapa, L., Matusik, W., Sueda, S., and
Agrawal, P. An end-to-end differentiable framework for
contact-aware robot design. *Robotics: Science and Sys-
tems*, 2021. Cited on page 1.

Zheng, L., Kumar, S., and Kochmann, D. M. Data-driven
topology optimization of spinodoid metamaterials with
seamlessly tunable anisotropy. *Computer Methods in
Applied Mechanics and Engineering*, 383:113894, 2021.
Cited on page 2.

A. Optimizer hyperparameters

Optimization hyperparameters were chosen to reflect good performance for each optimizer in each domain. We therefore performed sweeps for major hyperparameters of each optimizer for each domain, with those used for experiments in the paper shown in the table below.

CEM maintains a population of samples and uses these to estimate the mean μ and standard deviation σ of a Gaussian distribution over design parameters. To optimize μ and σ, it takes the top performing fraction, deemed the "elite portion," from the current step. The initial standard deviation of this distribution is given by "Initial σ", and the initial mean is set to 0. We found that for CEM, the population sample size had a significant effect on overall optimization quality (Figure A.1). Due to computational considerations, we picked the smallest value for this hyperparameter that performed within 1 standard deviation of the optimal sample size. Both the elite portion and initial σ parameters were chosen as the best performing values on a set of held-out random tasks for each domain.

For GD, we only performed a hyperparameter sweep over the learning rate, which was the only parameter to significantly affect performance. For the 2D FLUID TOOLS tasks, we introduced gradient clipping to eliminate the effect of rare gradient spikes over the course of optimization. However, not using gradient clipping still produced qualitatively and quantitatively similar results. For additional Adam parameters, we used the default values for the exponential decay rates that track the first and second moment of past gradients of $b_1 = 0.9$ and $b_2 = 0.999$ (Hessel et al., 2020).

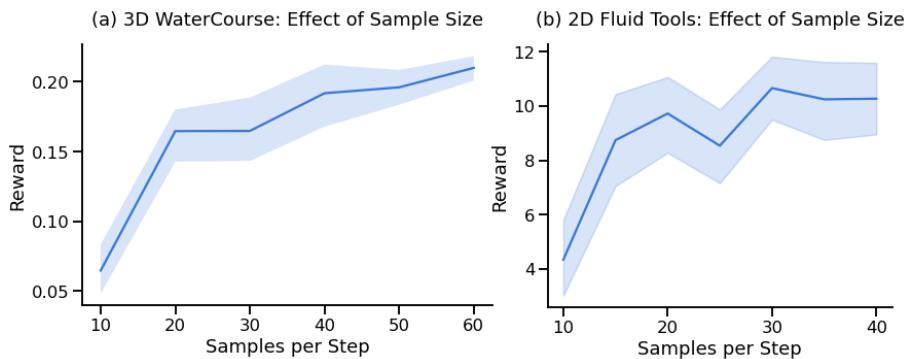| | 2D FLUID TOOLS | | | 3D WATERCOURSE | | AIRFOIL |
GD	*Contain*	*Ramp*	*Maze*	*Direction*	*Pools*	
Learning rate	0.005	0.005	0.01	0.01	0.01	0.01
Momentum term b_1	0.9	0.9	0.9	0.9	0.9	0.9
Momentum term b_2	0.999	0.999	0.999	0.999	0.999	0.999
Gradient clip	10	10	10	—	—	—
CEM						
Sampling size	20	20	20	40	40	—
Elite portion	0.1	0.1	0.1	0.1	0.1	—
Initial μ	0	0	0	0	0	—
Initial σ	0.5	0.5	1.5	0.1	0.1	—
Evolution smoothing	0.1	0.1	0.1	0.1	0.1	—
Optimization steps	1000	1000	1000	200	200	200

Figure A.1. For CEM, increasing population size, while more computationally expensive, can lead to improvements in performance. (a) In the 3D WATERCOURSE domain, CEM benefits from large sample sizes, although returns are diminishing for sizes beyond 40 (*Direction* task, 36 design parameters). (b) In the 2D FLUID TOOLS domain, CEM benefits from larger sample sizes, although returns are diminishing for sizes beyond 20 (*Contain* task, 40 design parameters).

B. Model architecture and training

For each task domain, we train a GNN for next-step prediction of the system state. For the domains considered in this paper, we unify the approaches of GNS (Sanchez-Gonzalez et al., 2020) and MESHGRAPHNETS (Pfaff et al., 2021): In the AIRFOIL domain, we encode/decode mesh nodes and mesh edges as a graph as described in the aerodynamics examples of MESHGRAPHNETS, while for particle-based fluids, edges are generated based on proximity as in GNS. In the case of 3D WATERCOURSE, both particles (fluid) and a mesh (the designed obstacle) are present; hence, edges are generated based on proximity (for fluid-fluid and fluid-obstacle interaction) or from the landscape mesh. As the landscape does not have any internal dynamics, we did not find it necessary to distinguish between world- and mesh edges, and use a single edge type.

Once encoded as a graph, the core model and training procedure is largely identical between GNS and MESHGRAPHNETS, and we refer to the above papers for full details on architecture and model training. Briefly, we use an Encode-Process-Decode GNN with 10 processor blocks. All edge and node functions are 2-layer MLPs of width 128, with ReLu activation and LayerNorm after each MLP block. The model is trained with Adam and a mini-batch size of 2, with training noise, for up to 10M steps. We implemented this model in JAX (Bradbury et al., 2018). In addition to the different encoding procedures for mesh vs. particle systems, the parameters for training noise and connectivity radius have to be set per-domain, to account for differences in particle size/mesh spacing. These details are described in Appendix C.

Gradient computation   In our experiments, we pass gradients through long model rollouts of up to 300 steps. As it is prohibitive to store all forward activations for the backwards pass, we use gradient checkpointing (Chen et al., 2016) to store activations only at the beginning of each step of the trajectory during the forward pass, and recompute the intermediate activations for each step as needed when the backwards pass walks the trajectory in reverse. Gradient calculation using this method has roughly 3 times the time cost of a pure forward simulation: forward dynamics have to be computed twice for each step, in addition to the computation of the backwards pass itself.

C. Task domains

C.1. 2D FLUID TOOLS

Tasks in 2D FLUID TOOLS are procedurally generated from templates specified in Table A.1. The simulation domain is a 2D box, with the lower left corner specified as $[0, 0]$, and upper right corner specified as $[1, 1]$. Fluid particles are initialized as a box of size *Initial fluid box* with bounding boxes given in format $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$. Certain task parameters were varied for ablation experiments in Figure 5 (rollout length, # joints (*Contain*), # tools (*Maze*)); Table A.1 contains default values used unless otherwise specified.

Design space   A "tool" in this task domain is a 2D curve composed of several line segments connected by joints. For a large number of joints, a tool can thus approximate a smooth curve (Figure A.2). Each task's design space consists of the relative joint angles controlling the tool's shape. We consider tasks with a single, multi-segment tool (*Contain*, *Ramp*) and a task with multiple, single-segment tools (*Maze*). For each tool, relative angles are calculated by moving from the anchor point on the left, along the tool segments to right, such that $\text{angle}_i = \text{angle}_{i-1} + \phi_{\text{joints}_i}$ for the ith joint from the anchor. We also experimented with two additional design space parameterizations: (1) jointly optimizing the joint angles and a global position offset $[x, y]$ for each tool, and (2) changing the parameterization of angles to be absolute (such that $\text{angle}_i = \phi_{\text{joints}_i}$ directly). We discuss the effects of these alternate parameterizations in Section D.2.

Simulation and objective   Both fluids and tools are represented as particles with different types, and simulated with the learned model for 150 steps (with the exception of the ablation experiment on rollout length). Scenes consist of $N = 100 \ldots 1000$ fluid particles. For ground-truth evaluation of the designs, we simulate particle dynamics with an MPM solver (Hu et al., 2018). Task reward is calculated using the Gaussian likelihood of the final particle positions after rollout ($\mathbf{u_v}$ from $\tilde{G}^{t_K}$). That is, for a task with reward parameterized with mean μ and spherical covariance σ ($\theta_R = [\mu, \sigma]$), the reward is calculated as

$$f_R := \text{mean}_v] \, \mathcal{N}(\mathbf{u_v}; \mu, \sigma) \, .$$

Contain   For this task, the center of the goal region μ is sampled uniformly from a rectangular reward region in the lower-middle section of the 1×1 simulation domain ($[0.4, 0.6] \times [0.2, 0.4]$). A tool protruding to the right is initially placed below the fluid rectangle. By optimizing a single tool's relative joint angles, successful solutions must "contain" the fluid in

	Contain	Ramp	Maze (nxn)
Environment size	1x1	1x1	1x1
Rollout length	150	150	150
Initial fluid box	[0.2, 0.5, 0.3, 0.6]	[0.2, 0.5, 0.3, 0.6]	[0.2, 0.75, 0.8, 0.8]
Reward sampling box	[0.4, 0.1, 0.6, 0.3]	[0.8, 0, 1, 0.2]	[0.1, 0.1, 0.9, 0.2]
Reward σ	0.1	0.1	0.1
Design parameter	joint angles	joint angles	rotation
# tools	1	1	n^2
# joint angles	16	16	1
Tool position (left)	[0.15, 0.35]	[0.15, 0.35]	—
Tool domain box (3x3)	—	—	[0.14, 0.3, 0.65, 0.6]
Tool domain box (4x4)	—	—	[0.14, 0.3, 0.71, 0.6]
Tool domain box (5x5)	—	—	[0.14, 0.3, 0.75, 0.6]
Tool domain box (6x6)	—	—	[0.14, 0.25, 0.77, 0.65]
Tool Length	0.8	0.8	—
Tool length (3x3)	—	—	0.72
Tool length (4x4)	—	—	0.64
Tool length (5x5)	—	—	0.65
Tool length (6x6)	—	—	0.63

Table A.1. Task Parameters for 2D FLUID TOOLS tasks. Boxes are described as $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$.

Figure A.2. Visualization of the design space parameterization for the 2D FLUID TOOLS task. Each red dot corresponds to the anchor points (*Contain* and *Ramp*) and center of rotation (*Maze*) being optimized.

the region by creating a cup or spoon.

Ramp   The fluid and tool are initialized as in *Contain*, and μ is sampled from a region lower and further to the right than in *Contain* ($[0.8, 1] \times [0, 0.2]$). By again optimizing a single tool's relative joint angles, successful solutions will create a "ramp" from the initial fluid position to the goal location in the bottom right.

Maze   The goal is sampled from a long region near the bottom of the domain ($[0.1, 0.9] \times [0.1, 0.2]$). By optimizing the rotation angles of a grid of rigid, linear tools, successful solutions will create a directed path from the top of the screen to the goal location at the bottom.

Model training   We trained the learned simulator on the WATERRAMPS datasets released by Sanchez-Gonzalez et al. (2020). This dataset consists of 1000 trajectories featuring a single large block of water falling on one to four randomized straight line segments (see image below for examples). Model architecture and hyperparameters are described in Appendix B, with a training noise scale of $6.7\,10^{-4}$ and connectivity radius of 0.015.
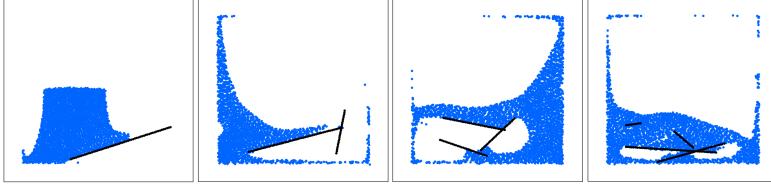

Figure A.3. Four examples of trajectories from the WATERRAMPS dataset released by (Sanchez-Gonzalez et al., 2020) used as training data for the supervised prediction model.

C.2. 3D WATERCOURSE

Design space   This domain has a design space ϕ_{map} of 625 parameters, which determine the y coordinate offset to nodes of a 25×25 square mesh centered at $\mathbf{c} = (0.5, 0.5, 0.5)$ in the simulation domain. While we could directly mapping the parameters to coordinates, we use the design function $y_i = \gamma_H \tanh(\phi_{\mathrm{map}_i})$ ($\gamma_H = 0.3$ for all tasks) to prevent trivial task solutions (i.e. obstacles which touch the floor).

Simulation   The simulation consists of an inflow pipe located at (-0.5, 1.0, 0.5) above the landscape which continually emits a stream of liquid, represented as particles. These particles are then redirected by the designed landscape, and finally removed once they hit the floor at $y = 0$. In our experiments we observed up to 2084 particles present in the scene at one time. We unroll the learned simulation model for trajectories of 50 time steps, and store the final particle positions $\mathbf{u_v}$, as well as the positions of removed particles that touched the floor at any point $\mathbf{u_v^D}$ to be passed to the reward function. Ground truth simulations for evaluation are performed by running the same setup with an SPH solver. We note that SPH requires very small simulation time steps, and performs $\approx 10^4$ internal steps for a trajectory of the same length.

Direction   In this task, we want to align the water stream with a given direction vector $\mathbf{d}$. We can formalize this using the reward function

$$f_{R_{\mathrm{dir}}} := \mathrm{mean}_v\left((\mathbf{u_v} - \mathbf{c}) \cdot \mathbf{d}\right) - \mathrm{std}_v\left((\mathbf{u_v} - \mathbf{c}) \cdot \mathbf{d}_\perp\right) - \gamma_R \, \mathrm{mean}(\nabla \phi_{\mathrm{map}})$$

where $\mathbf{d}_\perp$ is orthogonal to $\mathbf{d}$. The first term aligns the direction of the particle relative to the domain center, and the second term concentrates the stream. The last term is a smoothness regularizer on the design landscape, which prefers smooth solutions ($\gamma_R = 300$ for both tasks). Absolute reward numbers for this task can be positive or negative, hence we report the normalized reward $f_{R_{\mathrm{dir}}} - f_{R_{\mathrm{dir}}}^{\mathrm{initial}}$, i.e. an unchanged initial design corresponds to a zero reward, to make the scores easier to interpret.

Rewards can be in 8 different directions, spaced between 0 and $180 \deg$. We collapse across directions for reporting reward means and confidence intervals for each optimizer.

2 Pools **and** ***3 Pools***   In these tasks, we define two and three pools, respectively, with center $\mu_{\mathbf{p}}$ on the floor. For each particle which has hit the floor, we assign it to its closest pool $\hat{\mu}_{\mathbf{p}}$, and define the reward as the Gaussian probability under $\hat{\mu}_{\mathbf{p}}$, i.e.

$$f_{R_{\mathrm{pools}}} := \mathrm{mean}_v(\mathcal{N}(\mathbf{u_v^D}; \hat{\mu}_{\mathbf{p}}, \sigma)) - \gamma_R \, \mathrm{mean}(\nabla \phi_{\mathrm{map}})$$

with $\sigma = 0.4$ and a regularization term as above.

To showcase different ways of splitting the water stream, we consider one positioning of the pools for the two pool case, and two for the three pool case. In the two pool case, pools are placed at $[1.49, -0.35]$ and $[1.49, 1.35]$. In the three pool case, pools are placed either at $[1.6, -0.45]$, $[1.85, 0.5]$, and $[1.6, 1.45]$, or at $[0.5, -0.5]$, $[1.7, 0.5]$, and $[0.55, 1.5]$. These were selected to ensure the task was solveable – pools directly beneath the landscape, or too far away from the landscape, would not be reachable even with dramatically warped surfaces.

Model training   We trained a model on next-step prediction of particle positions, on a dataset of 1000 trajectories of water particles interacting with a randomized obstacle plane (random rotations and sine-wave deformations of the planar obstacle surface). The data was generated using the SPH simulator SPlisHSPlasH (Bender & Koschier, 2015). The noise scale is set to 0.003 and a connectivity radius of 0.01 to account for the different particle radius of the 3D SPH simulation compared to 2D MPM. All other architectural and hyperparameters are as described in Appendix B.

C.3. AIRFOIL

The airfoil optimization task is modeled similarly to the NACA0012 aerodynamic shape optimization configuration for incompressible flow for the DAFoam solver (see details here), to make it easier to compare design solutions to this solver.

Design space   The design space consists of the y-coordinate of 10 control points (see Figure 4a). Moving these control points deforms both the airfoil, and the simulation mesh surrounding it. The airfoil shape is deformed using B-spline interpolation as described by Reid (2021), and the mesh is deformed using IDWarp (Secco et al., 2021). We thus define a design function $G^{t0} = f_D(\phi_{\text{ctrl}}, G_\alpha)$ which takes an initial, undeformed airfoil mesh (we use the standard NACA0012 airfoil), encoded as a graph G_α, as well as the control point position ϕ_{ctrl} as input. It returns the graph of the deformed airfoil mesh G^{t_0} to be passed to the simulator. We note that the coefficients for spline interpolation and mesh warping can be precomputed for a given initial mesh, making it easy to define a differentiable function to use for design optimization.

Simulation   Given the initial mesh, as well as simulation parameters, the simulator or learned model predict the steady-state incompressible airflow around the wing, sampled on each of the 4158 nodes on the simulation mesh. The entire simulation domain and an example prediction of the pressure field are shown in Figure A.4a,b. For drag minimization we require predictions of the pressure field p, as well as the effective Reynolds stress ρ_{eff} at each mesh node, i.e. $\mathbf{q}_v = (p, \rho_{eff})$. Unlike the other domains in this paper, this is a single-step prediction task, and model rollouts are of length one. For this task, we consider an inflow speed of 0.1 mach, under an $5.1°$ angle of attack.

Task objective   The task reward is defined as $f_R := -C_D - \gamma_L ||C_L - C_{L0}||^2 - \gamma_A\, a(\phi_{\text{ctrl}})$, i.e. we minimize the drag coefficient C_D under soft constraints of unchanged lift C_L and a wing area a of 1-3 times the initial area. We use $\gamma_L = 10, \gamma_A = 1$, and a tanh nonlinearity to enforce the volume inequality. Lift and drag can be computed from the simulation output p, ρ_{eff} by integration around the airfoil, see e.g. (Ladson, 1988). We report the normalized reward $f_R - f_R^{\text{initial}}$ such that the initial, undeformed wing design corresponds to a zero reward.

Model training   We trained a model to predict p, ρ_{eff} on a dataset of 10000 randomized airfoil meshes, simulated with OpenFoam (OpenCFD Ltd, 2021). For training ensemble models, this dataset is split into 5 non-overlapping blocks, and a separate model is trained on each section. Since this is a steady-state prediction task, information needs to propagate further at each model evaluation. We therefore use twice-repeated processor blocks with shared parameters, i.e. the model performs 20 message passing steps, with 10 blocks of learnable parameters. We found that this increases accuracy in the one-step setup by being able to pass messages further across the mesh. Training noise is often cited for stability over long rollouts, but even in this one-step setting, training noise and data variation can be useful. To increase robustness to unseen wing configurations, we varied the grid resolution between 1000-10000 nodes for each sample in the training set, and added training noise to the input mesh coordinates. We use a normal noise distribution with the scale of 1% of the average edge lengths surrounding the node noise is applied to. All other aspects of model architecture and training procedure are as described in Appendix B.

D. Further results

D.1. Model accuracy

In order for a learned simulator to be useful for design, it must be sufficiently accurate in the forward direction. We study this question directly for each of the domains (3D WATERCOURSE, 2D FLUID TOOLS and AIRFOIL) by examining the magnitude of the error between the model predictions of reward for the discovered designs, and the ground truth reward for those designs. The results for each domain are shown in Figure A.5.

Broadly, the learned model very successfully mimics the ground truth simulator in reward prediction across all three domains. The accuracy for AIRFOIL is within a single standard deviation across all ensemble sizes, while the predictions in 3D
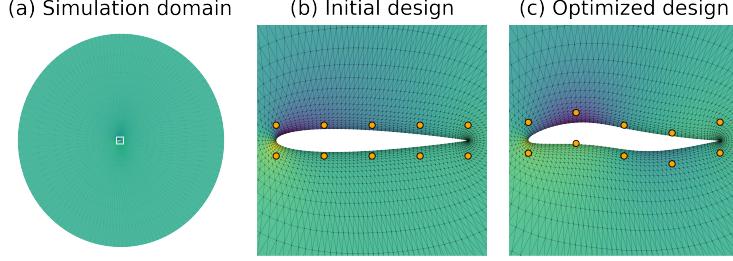
Figure A.4. (a) Aerodynamics are computed on a large 4158 node mesh centered around the airfoil, with the closeup regions around the airfoil in (b, c) marked as a white square in the center. (b, c) Pressure predictions and control points (orange) for the initial and final optimized wing design (Ensemble-5 model).

WATERCOURSE match very closely for both the high performing designs (GD-M) and low performing designs (CEM-M).

However, we do notice some discrepancies in the predicted and ground truth reward for the 2D FLUID TOOLS domain, particularly the *Maze* task. As mentioned in the main text, the ground truth solver sometimes produces unrealistic rollouts for this domain (see Figure A.9), with fluid particles becoming stuck between the different tools. Despite this issue, we find that the model is sufficiently similar to the ground truth to produce designs that still achieve high reward overall.

Figure A.5. (a) For the 3D WATERCOURSE domain, reward predicted by the learned model (GD-M eval w/ M, CEM-M eval w/ M) is very close to the ground-truth simulator evaluation (GD-M, CEM-M) for all tasks. (b) This is also true for the 2D FLUID TOOLS domain, though the reward is slightly overestimated when using the model. This effect is amplified in *Maze*, where the ground-truth dynamics sometimes struggles to correctly simulate "sticky" bottlenecks (see Figure A.9). (c) Model predictions of drag (GD-M eval w/ M) are relatively close to the ground-truth simulator evaluation (GD-M), particularly for larger ensemble sizes.

D.2. Effects of design parameterization

In this section, we study how different parameterization choices for the design space affect both gradient-based and sampling-based optimizers.

First, in the 3D WATERCOURSE domain, we investigate a parameterization of the design space that uses interpolation to minimize the number of control points on the 2D heightfield (Figure A.6). Control points are placed evenly across the grid, and bi-linearly interpolated onto the 25×25 mesh. We vary the number of control points from 2×2 up to 14×14, and find that while CEM-M performs similarly to GD-M when very few control points are allowed, its performance quickly drops as more control points are added.
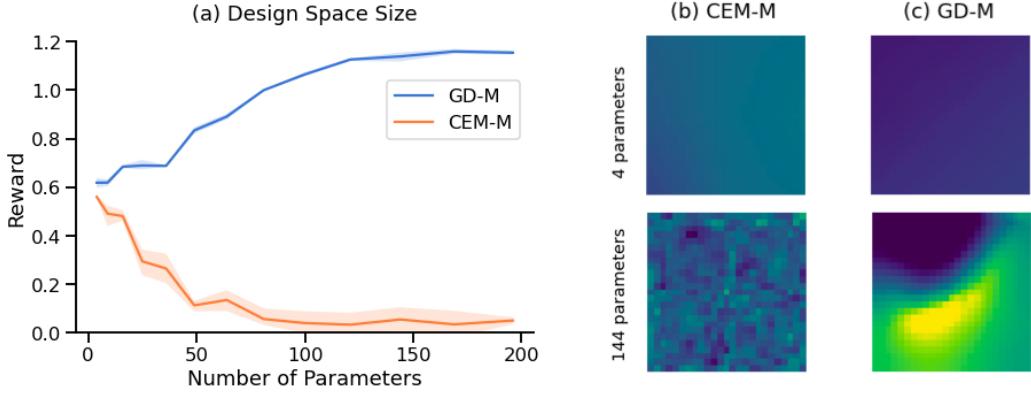
Figure A.6. Performance on the 3D WATERCOURSE *Direction* task with variable design space resolution: GD performs well for large design spaces, while CEM performance quickly drops with increased number of design parameters. (b) Examples of CEM designs at two different design space resolutions. (C) Examples of GD designs at two different design space resolutions.

Second, in the 2D FLUID TOOLS domain, we investigate what happens when we change the design space to use *absolute* joint angles rather than relative ones. When using relative joint angles, changes to joints near the tool's pivot (left side) affect the global properties of the tool. We hypothesized that this could be selectively benefiting the sampling-based approaches, as this makes the effective design space much lower dimensional. We therefore change the design space to be absolute, with a tool's joint angles calculated directly: $angle_i = \phi_i$.

As hypothesized, this change does dramatically decrease the performance of the sampling-based technique (see Figure A.7). Perhaps more surprisingly, the gradient-based optimizer is almost completely unaffected by this reparameterization. While the qualitative solutions it finds differ (with tools now containing "kinks" to prevent the motion of the fluid rather than curves, Figure A.7 top), the overall reward achieved is similar.

Figure A.7. **(a)** Example solutions for each optimizer across the *Contain* and *Ramp* tasks when optimizing over *Relative* vs *Absolute* angles. **(b)** Mean reward with 95% confidence intervals obtained by each optimizer across the *Contain* and *Ramp* tasks when optimizing over *Relative* vs *Absolute* angles.

D.2.1. FAILURE MODES OF GRADIENT DESCENT

Other parameterizations of the design space can badly affect the performance of gradient-based optimizers. In particular, gradient-based optimizers suffer when there are regions of zero gradients. In the AIRFOIL and 3D WATERCOURSE domains, this is not normally a problem, as the design always interacts with the physical system on which reward is being measured. But in the 2D FLUID TOOLS domain, we can manipulate this.

In particular, for this experiment we changed the design space for 2D FLUID TOOLS to include a global position offset $[x, y]$ for the tool. Making this simple change often has no effect on the discovered designs, but occasionally the gradient-based optimization procedure can move the tool such that it no longer interacts with the fluid (Figure A.8). Once the tool has been moved out of the range of the fluid, there is no longer any way to affect the reward, and therefore there is no gradient signal to recover. To overcome this problem, future work would need to consider more sophisticated hybrid optimization techniques (Toussaint et al., 2018).
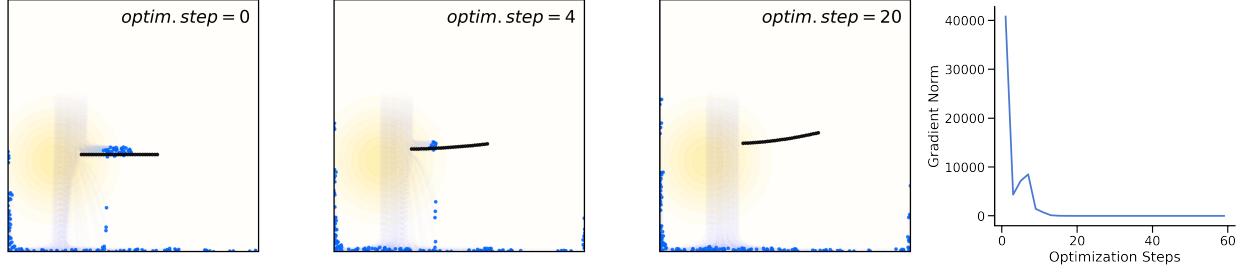
Figure A.8. Failure mode of the GD optimizer: in some instances where the translation of the tool is included in the design, the tool may end up outside of the scope of the fluid. In this cases, the optimization can no longer recover as it will get zero gradients from there on.

D.3. Failure modes of the MPM solver

One of the advantages of using a learned simulator over a classic simulator is learned simulators can be trained in regions of the state and action space that are known to exhibit regularized, smooth behaviors. For example, as mentioned in Section D.1, the MPM solver (Hu et al., 2018) we use for evaluation in 2D FLUID TOOLS shows surprising irregularities with "sticking" behavior when there are a large number of different tools. In the *Maze* task, this is particularly prevalent, as fluids often become stuck stochastically in some funnels but not others of similar sizes (Figure A.9).

Since the learned simulator was trained on much simpler scenarios where this effect is not observed, it only learns the smooth behavior of the fluid's movement, which makes the resulting trajectories look more realistic. This may enable better generalization to real world scenarios.
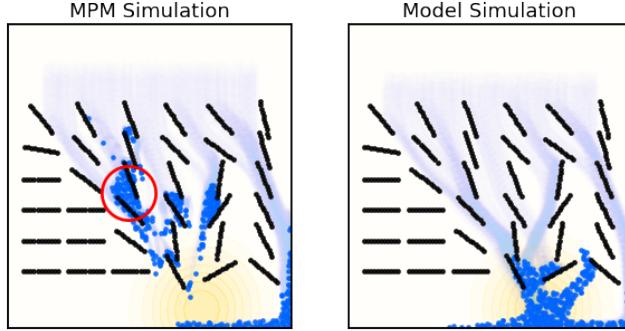

Figure A.9. Left: MPM simulation (Hu et al., 2018) of a problem with many separate solid objects. As highlighted in the red circle, the MPM solver struggles with water movement between obstacles, often creating artificially sticky bottlenecks. Right: Learned model rollout for the same setup. The model rollout looks significantly more plausible, without any "stickiness" artifacts. Please see https://sites.google.com/view/optimizing-designs for videos demonstrating this effect clearly.

D.3.1. FURTHER DESIGNS FOUND IN 2D FLUID TOOLS

In the figures below, we demonstrate the range of found solutions for different solvers in tasks in the 2D FLUID TOOLS domain.
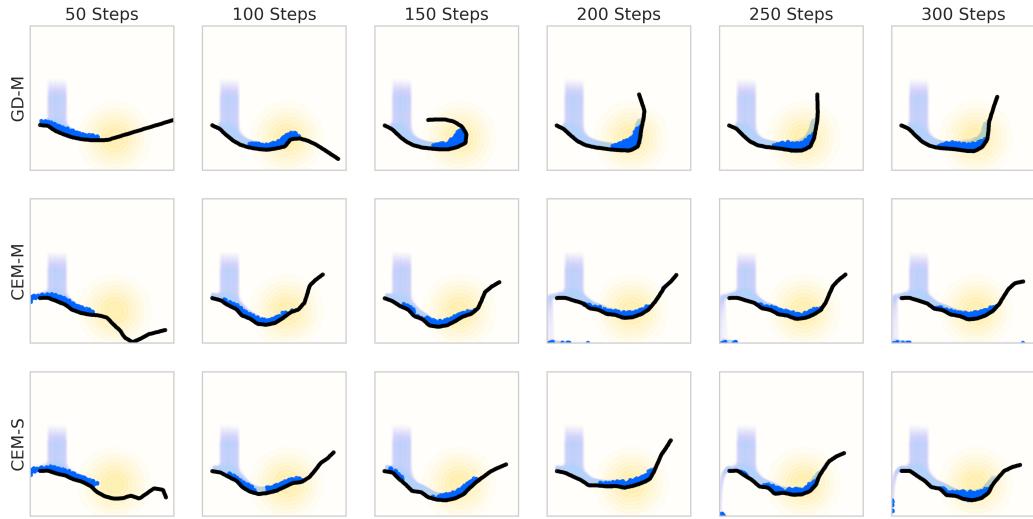
Figure A.10. Example solutions for each optimizer across the range of rollout lengths sampled for *Contain* in Figure 5.
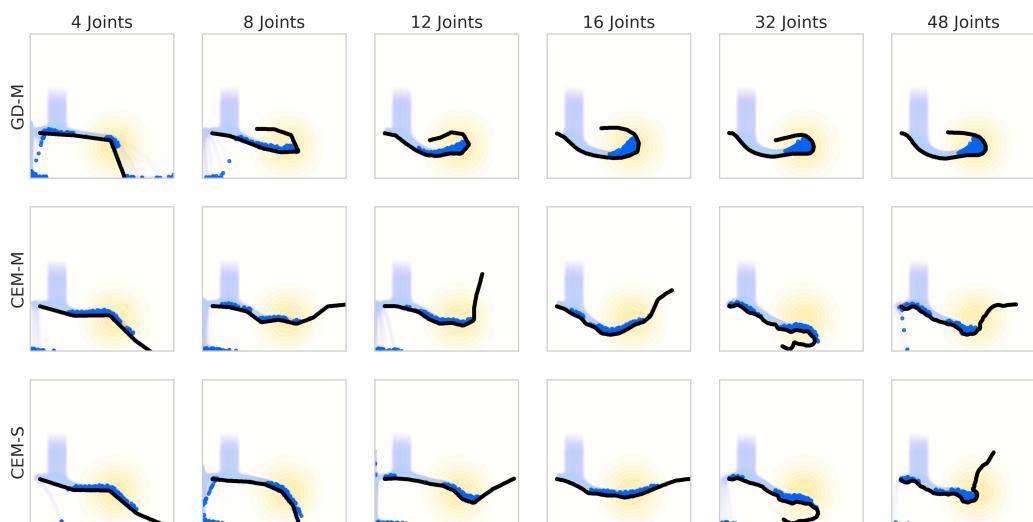

Figure A.11. Example solutions for each optimizer across the range of joint angle numbers sampled for *Contain* in Figure 5.
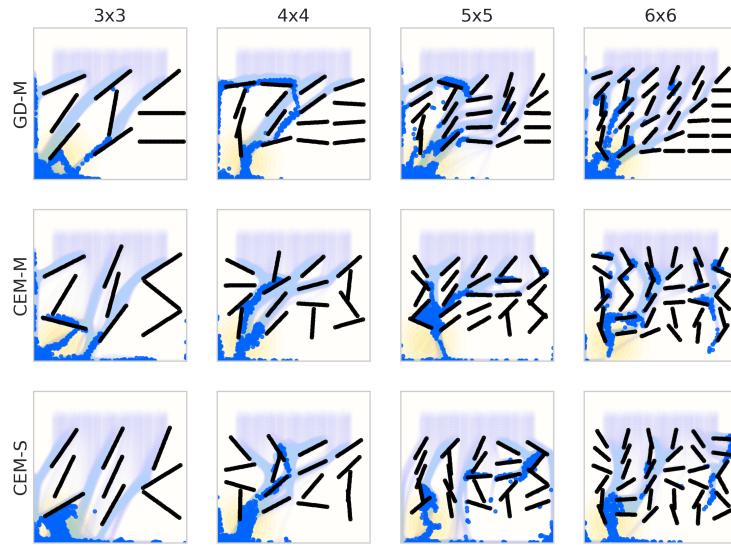
Figure A.12. Example solutions for each optimizer across the range of grid sizes sampled for *Maze* in Figure 5.
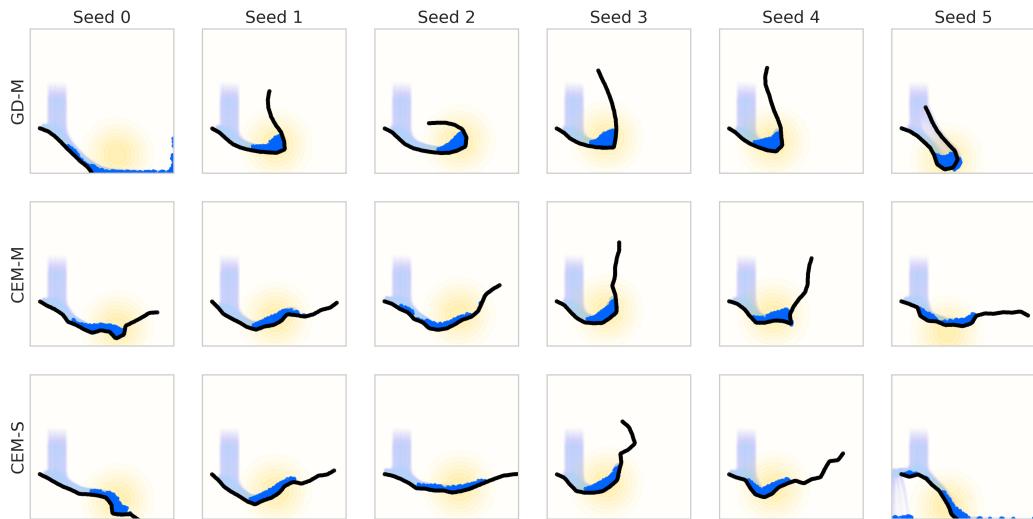

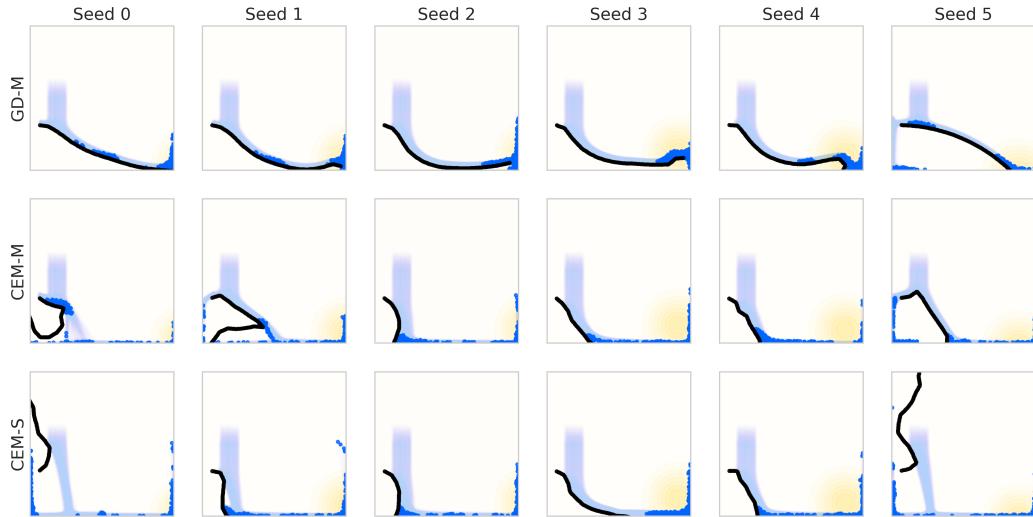*Figure A.13.* Example solutions on *Contain* task for each optimizer across 6 random seeds.

Figure A.14. Example solutions on *Ramp* task for each optimizer across 6 random seeds.
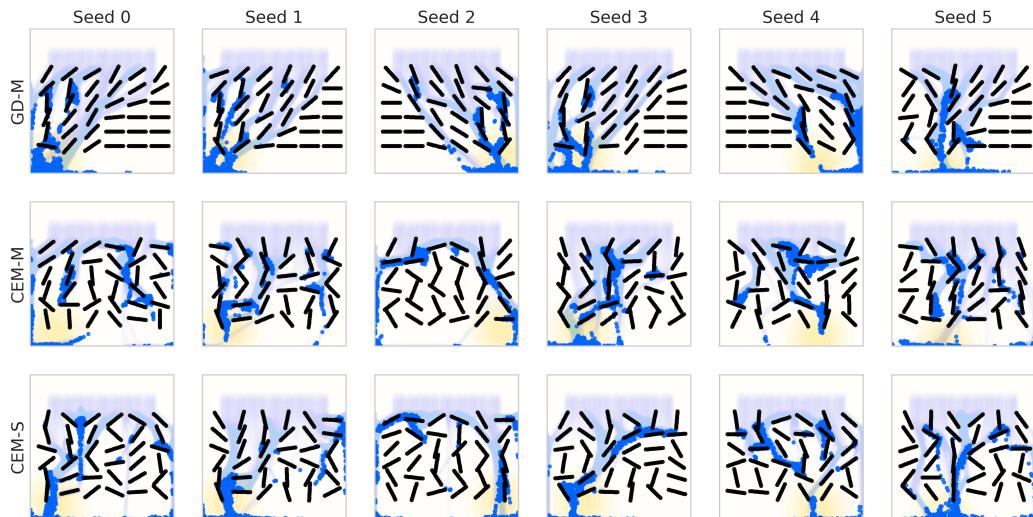

Figure A.15. Example solutions on *Maze* for each optimizer across 6 random seeds.