Inverse Design for Fluid-Structure Interactions using Graph Network Simulators

Anonymous Author(s)
Affiliation
Address
email

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 using machine learning has tremendous promise, existing methods are often limited by the task-dependent distributions they were exposed to during training. Here we showcase a *task-agnostic* approach to inverse design, by combining general-purpose graph network simulators with gradient-based design optimization. This constitutes a simple, fast, and reusable approach that 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 fluid-structure interaction domains.

1 Introduction

Automatically designing objects to exhibit a desired property—often referred to as *inverse design*—has the potential to transform science and engineering, including aerodynamics [27], material design [14], optics [21], and robotics [35, 75]. Despite its promise, widespread practice of inverse design has been limited by the availability of fast, general-purpose forward models. Traditionally, many methods rely on specialized "classical" solvers, which are handcrafted to simulate a particular physical process. However, these models can sometimes be slow, may not provide gradients, or in certain domains, might even be unknown [23].

To mitigate these issues, machine learning is playing an increasingly important role by substituting all or part of the inverse design pipeline. "End-to-end" methods can bypass expensive simulation and data collection by learning a mapping directly from design parameters (such as the shape of an airplane wing) to a task objective (such as the observed drag) [59, 76, 19]. Learned generative models or policies can propose candidate design parameters for a given task [52, 17, 19, 24, 12, 47]. These types of methods can be used to solve narrowly defined tasks more efficiently than traditional methods, but come with significant limitations: they require access to a dataset of example designs and tasks for training, and often cannot generalize to tasks or designs outside this distribution [71, 77].

In this work, we focus on a *task-agnostic* approach to inverse design, by employing learned simulator models pre-trained on dynamics with *no* access to the task objective or design space during training,

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

and that, once trained, can be used to solve many new design tasks. This approach, however, requires models with sufficient long-term stability and strong generalization, which commonly applied black-box model architectures such as CNNs often lack [45, 69, 72, 4].

A class of learned physics simulators based on graph neural networks (GNNs) [57, 63] could be a promising candidate for supporting inverse design, as they have shown success in general-purpose physical prediction, exhibiting high accuracy and generalization for a broad range of physics tasks. However, these models have so far mostly been tested for forward prediction alone. To be useful in a design setting, they must also be smooth and robust, as optimizers will exploit unreliable regions in the state space [53]. Moreover, to use scalable gradient-based optimization techniques, simulation models must additionally exhibit gradient stability over extremely large state and action spaces, and over hundreds of time-steps. It is unclear if graph network simulators exhibit these properties.

In this paper, we investigate whether learned GNN simulators can support effective inverse design. We pretrain a GNN simulator on a task-independent dynamics dataset, and at test time perform gradient-based optimization for zero-shot design across a variety of tasks. We perform a large set of experiments and find that (1) the learned simulator model is accurate and flexible enough to support a wide variety of design tasks; (2) it generalizes well enough to support design tasks far outside the training regime; (3) gradients from the learned model turn out to be very useful for inverse design, with gradient-based optimization strongly outperforming a variety of sampling-based optimizers; (4) and surprisingly, gradients remain stable even when rolling the model out for hundreds of steps. In experiments on high-dimensional fluid manipulation (2D FLUID TOOLS and 3D WATERCOURSE) we demonstrate that gradient-based optimization on learned models can find high-quality designs over hundreds of time steps, through states with thousands of particles, in tasks with up to 625 design parameters. The same model could match performance of a specialized solver on aerodynamics design (AIRFOIL) with a much simpler setup, highlighting the flexibility and precision of this approach.

2 Related Work

Modern design techniques rely on physical simulators so that designs can be evaluated virtually before being tested physically. **Solving inverse design problems with simulators** has been studied in a variety of disciplines, including nanophotonics [54], material science [25], mechanical design [22], and aerodynamics [2, 60]. Many of these rely on black-box optimization, including active and adaptive sampling methods [68, 3]. In recent years, differentiable simulators [29, 44, 64] have garnered attention, as they allow for more sample-efficient gradient-based optimization. However, like classical solvers, they are typically narrow in application scope, as many simulation techniques are hard to express as a differentiable program (e.g. constraint dynamics and multiphysics coupling). Our work aims to lessen this burden by learning these simulators from data.

Machine learning for inverse design replaces or augments the simulation model and/or the optimization procedure with learned modules [e.g. 16, 19, 34, 28, 37, 42, 48, 50, 52, 59, 66, 78]. Generative approaches enable efficient optimization by learning to sample more likely designs (thus calling an expensive simulator fewer times). End-to-end predictive approaches learn a surrogate model that maps designs to task rewards directly (replacing the simulator entirely). This permits not only sampling-based optimization, but also gradient-based techniques when the surrogate model is differentiable [59]. However, surrogate models often generalize poorly to designs that lie outside of their training dataset, so many ML approaches for design must keep the optimization procedure close to the training dataset [12, 30], or introduce alternative objectives for the surrogate model to increase robustness [71, 77]. In all cases, these methods require access to datasets with (design, task reward) pairs on the specific task, making zero-shot generalization to new tasks impossible. Instead, we propose to learn the underlying physical dynamics *independent of the design task*, allowing both the design space and task to be specified at design time.

In this way, our approach is related to methods in **model-based reinforcement learning** (MBRL), which learns action-conditioned transition models to support flexible downstream planning and control. However, learned simulators are rarely accurate, stable, and generalizable enough to support better downstream task performance [36, 53]. Techniques from MBRL need to mitigate model error either by learning a global, task-conditioned *policy* during model learning [45] (which then cannot be adapted to novel tasks on-the-fly), or by *re-planning* at every timestep to avoid error accumulation [15]. Because of such sensitivities, these methods have only been demonstrated in

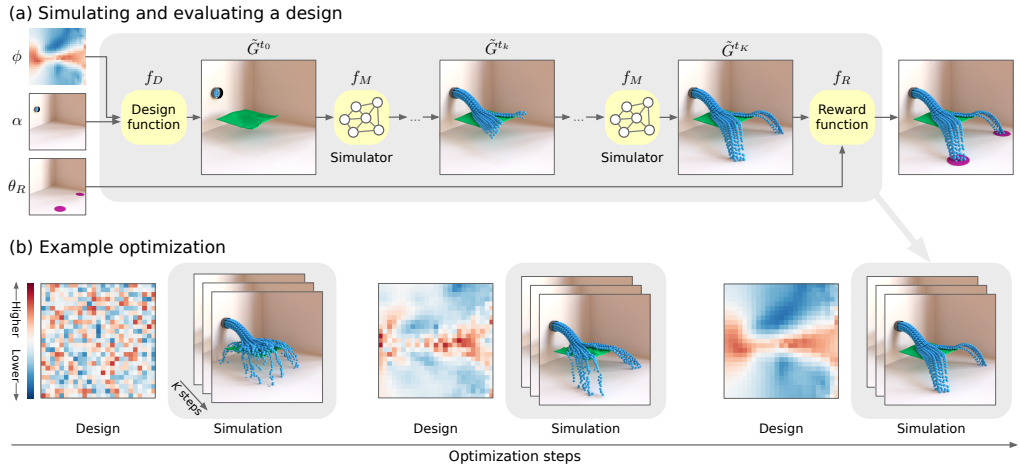

Figure 1: Optimizing a physical design. Here, the goal is to direct a stream of water (shown in blue) into two "pools" (shown in purple) by designing a "landscape" (shown in green) parameterized as a 2D height field. (a) The simulation pipeline takes in a design ϕ and initial conditions α and uses the 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.

relatively small-scale problems, for example with (up to 21 degrees of freedom DOF [10, 62, 73]) or more complex domains with small action spaces (6 DOF [51]), unlike the thousands of state dimensions and hundreds of action dimensions and timesteps we consider here.

We focus on **graph neural networks** as our simulator model, as they have recently demonstrated high accuracy, stability, domain-generality, and generalization performance in the forwards direction for physics domains [7, 57, 63]. However, it is not known whether these simulators can now support gradient-based, high-dimensional inverse design which demands high accuracy, well-behaved gradients, long-term rollout stability, and generalization beyond the training data. Our experimental investigations show that they can, and that particular architectural choices are critical to this success.

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.

Learned simulators To demonstrate the utility of learned simulators for finding physical designs, we rely on the recently developed MESHGRAPHNETS model [57], which is an extension of the Graph Network Simulator (GNS) model for particle simulation [63]. MESHGRAPHNETS is a type of message-passing graph neural network (GNN) that performs both edge and node updates [6, 32], and which was designed specifically for physics simulation by enforcing locality and translation equivariance. We briefly summarize the learned simulator here, 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 MESHGRAPHNETS) 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). The model is constrained to perform local computation, by performing message passing over these edges, and to perform equivariant computations by encoding position as the relative displacement and distance between neighboring particles. Intuitively, this allows it to generalize well to scenarios where the local interactions are the same but the global arrangement of particles changes. See Appendix B for model implementation details, and Appendix C for domain-specific details.

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 MESHGRAPHNETS, 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. To improve stability for longer rollouts, random Gaussian noise was added to the inputs during training.

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 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 reward function 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). Reward function parameters θ_R are the means and standard deviations of these pools.

We define the 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 by maximizing $J_M(\phi) := f_R(f_M^{(K)}(f_D(\phi, \alpha)); \theta_R)$. Crucially, multiple reward functions can be optimized with the same model f_M.

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 [10]. We use the Adam optimizer [46] 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) [61], a gradient-free sampling-based technique that is popular in model-based control [20, 73]. 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 iteration (depending on the population size), whereas Adam requires only a single evaluation per iteration.

Across our design tasks (Section 4) we compared: gradient descent with Adam on the learned simulator model (**GD-M**), CEM with the learned simulator model (**CEM-M**), and CEM with the ground-truth simulator (**CEM-S**). We compare to further optimizer baselines, such as Bayesian Optimization [68] and CMA-ES [38] in the Appendix (Section D.2). 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** [40], 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

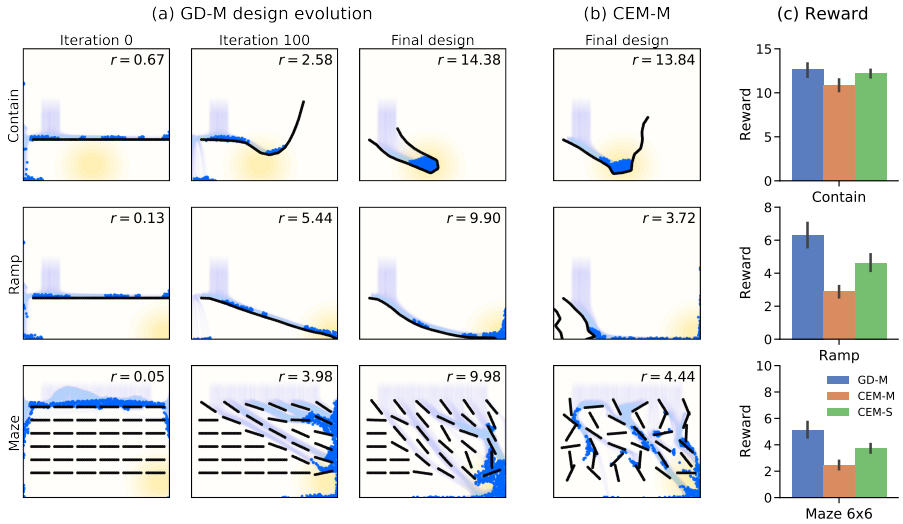

Figure 2: 2D FLUID TOOLS results. The state spaces consist of 10^2–10^3 particles and the design spaces of 16–36 parameters. (a) Evolution of designs found by GD-M during optimization for each 2D FLUID TOOLS task. Visualizations correspond to simulations of the designs under f_S. The design is shown 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.

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 formulate a set of design tasks across three different physical domains with high-dimensional state spaces and 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 in a task-agnostic way. During optimization, all the design tasks on each domain share the same pre-trained model.

2D FLUID TOOLS Inspired by existing 2D physical reasoning benchmarks [1, 5], 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 in [63] which uses the solver in [43] 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 out of distribution for the learned model.

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

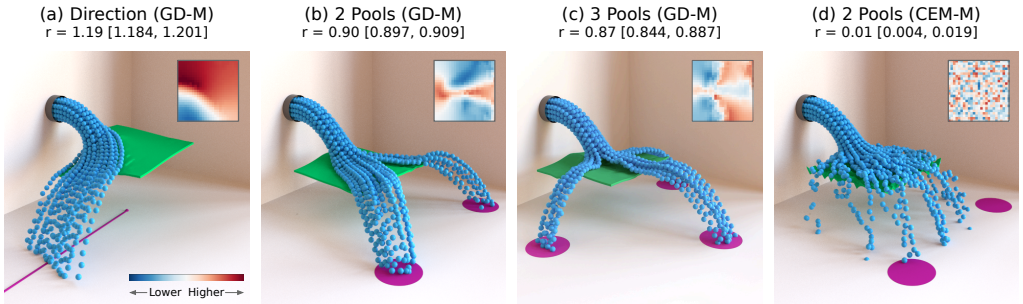

Figure 3: 3D WATERCOURSE results found with GD-M or CEM-M and evaluated with f_S. Simulations use up to 2000 particles and 625 design parameters. The heightmap of a 2D landscape is optimized to redirect the fluid towards the purple targets; birds eye views of heightmaps are shown in the upper right corner 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).

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 [8].

AIRFOIL Shape optimization in aerodynamics is one area where gradient-based optimization is routinely applied using traditional simulators [13]. Here, we consider the well-studied task of drag optimization of a 2D airfoil profile (Figure 5, 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 task 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 with the OpenFOAM solver [55].

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 hyperparameters in 2D FLUID TOOLS and 3D WATERCOURSE, 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 for details). Importantly, all the pre-trained models are shared across the multiple design tasks for each domain.

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). Here we discuss these results, and compare the capabilities of gradient descent with learned simulators over classical simulators and sampling-based optimization techniques.

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 [9], or domain shift [36, 71]. However, as the following results show, GD-M produced high-quality designs across all three domains.

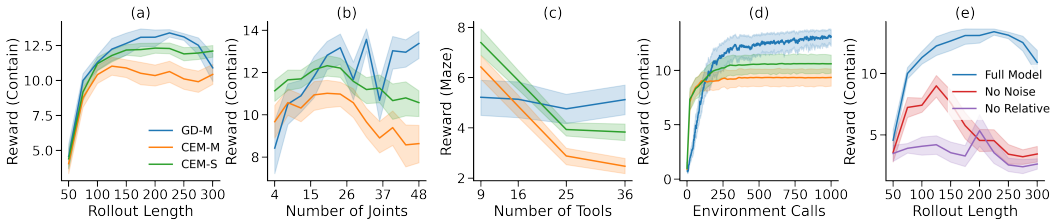

Figure 4: Ablation experiments on the *Contain* and *Maze* tasks. (a) Performance of all optimizers increase with rollout length; GD-M performance starts to deteriorate around step 225. (b) In *Contain*, CEM performance drops when increasing the number of joints above 24, while GD-M remains stable. (c) 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. (e) Model choice matters: good performance on GD-M is enabled by key architectural choices of the learned model, such as training noise and translational equivariance.

Figure 2a shows qualitative results for 2D FLUID TOOLS (Section 4), 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), 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). By comparison, CEM-M cannot solve any of these tasks, with performance 30–85$\times$ worse than GD-M. In the Appendix (Section D.2) we demonstrate that these results also hold for other optimizer choices (Bayesian Optimization, CMA-ES).

In AIRFOIL (Figure 5b), 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. 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). Importantly, DAFoam's solver and optimizer are highly specialized for the particular task of airfoil design, while our approach is more general-purpose, requiring only trajectory data for training and a generic gradient-based optimizer.

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 4a and A.11). 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 [69, 72]. 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 4a), indicating that the learned simulator's accuracy and gradients remain stable for a surprisingly long time. Across the episode lengths evaluated, 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 small errors 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 [51].

We hypothesize that a key reason for the success of GD-M is the addition of noise in training the model f_M, which promotes stability on the forward pass and smoother gradients. We test this by considering a model variant trained without noise (Figure 4e, red). We observe much worse design quality compared to the full model, particularly for longer rollouts.

Generalization Deep networks often struggle to generalize far from their training data [31]. 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 generalize well in the setting studied here. As noted in Section 4, the learned simulator for 2D FLUID TOOLS was trained on a pre-existing, highly simplified dataset

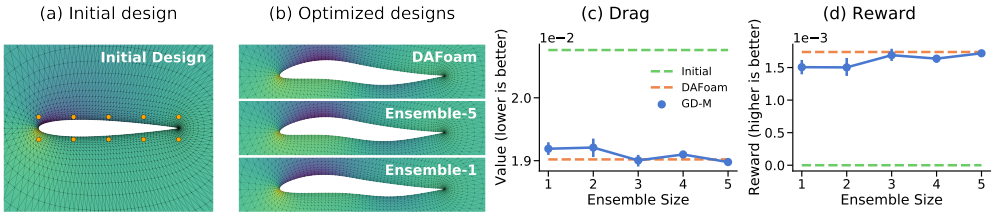

Figure 5: AIRFOIL results. (a) An initial airfoil design is warped by moving 10 control points (orange dots). The physics model simulates the resulting aerodynamics on a 4158 node mesh, based on which lift and drag are computed. (b) For the task of finding a minimum-drag configuration under constant 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.

where only one to four straight line segments interact with a fluid (Appendix C). However, it still can support the design tasks studied here, which involve highly articulated, curved obstacles (*Contain*, *Ramp*) or a larger number of obstacles (*Maze*), without further finetuning (Figure 2).

This generalization capability emerges due to certain design choices of the simulator model f_M. Specifically, the GNN operates on local neighborhoods of particles, and particle coordinates are represented in a relative manner, which encourages learning local rules that can transfer to very different setups. We tested a model variant which breaks this property by encoding *absolute*, instead of *relative* particles coordinates (Figure 4e, purple). We find that this variant can no longer robustly simulate out-of-domain designs, leading to worse performance at any rollout length.

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.10). 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.

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 5b, 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 [20], 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 5b-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.

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 4b), 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 4c), 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.7). 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.

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 63], 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 find that state-of-the-art learned, differentiable physics simulators can be used 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 [1, 56, 67]. 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.9). 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 [17, 70]. 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

[1] Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 2020.

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

[3] Christof Angermüller, David Belanger, Andreea Gane, Zelda Mariet, David Dohan, Kevin Murphy, Lucy J. Colwell, and D. Sculley. Population-based black-box optimization for biological sequence design. *CoRR*, abs/2006.03227, 2020. URL `https://arxiv.org/abs/2006.03227`.

[4] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *CoRR*, abs/1805.12177, 2018. URL `http://arxiv.org/abs/1805.12177`.

[5] Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. Phyre: A new benchmark for physical reasoning. *Advances in Neural Information Processing Systems*, 32:5082–5093, 2019.

[6] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[7] Filipe de Avila Belbute-Peres, Thomas D. Economon, and J. Zico Kolter. 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.

[8] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 147–155, 2015.

[9] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[10] Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. Model-predictive control via cross-entropy and gradient-based optimization. In *Learning for Dynamics and Control*, pages 277–286. PMLR, 2020.

[11] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[12] David H. Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design, 2019. URL `https://arxiv.org/abs/1901.10060`.

[13] Howard P Buckley, Beckett Y Zhou, and David W Zingg. Airfoil optimization using practical aerodynamic design requirements. *Journal of Aircraft*, 47(5):1707–1719, 2010.

[14] Keith T Butler, Jarvist M Frost, Jonathan M Skelton, Katrine L Svane, and Aron Walsh. Computational materials design of crystalline solids. *Chemical Society Reviews*, 45(22):6138–6146, 2016.

[15] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

[16] Adithya Challapalli, Dhrumil Patel, and Gouqiang Li. Inverse machine learning framework for optimizing lightweight metamaterials. *Materials & Design*, 208:109937, 2021.

[17] Chun-Teh Chen and Grace X. Gu. Generative deep neural networks for inverse materials design using backpropagation and active learning. *Advanced Science*, 7(5):1902607, 2020.

[18] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[19] Thomas Christensen, Charlotte Loh, Stjepan Picek, Domagoj Jakobović, Li Jing, Sophie Fisher, Vladimir Ceperic, John D. Joannopoulos, and Marin Soljačić. Predictive and generative machine learning models for photonic crystals. *Nanophotonics*, 9(13):4183–4192, 2020.

[20] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

[21] Shane Colburn and Arka Majumdar. Inverse design and flexible parameterization of meta-optics using algorithmic differentiation. *Communications Physics*, 4(1):1–11, 2021.

[22] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Trans. Graph.*, 32(4), jul 2013.

[23] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.

[24] Yabo Dan, Yong Zhao, Xiang Li, Shaobo Li, Ming Hu, and Jianjun Hu. Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials. *Nature Computational Materials*, 6(1):84, 2020.

[25] Marjolein Dijkstra and Erik Luijten. From predictive modelling to machine learning and reverse engineering of colloidal self-assembly. *Nature Materials*, 20(6):762–773, 2021.

[26] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[27] Richard Eppler. *Airfoil design and data*. Springer Science & Business Media, 2012.

[28] Antonio Elia Forte, Paul Z. Hanakata, Lishuai Jin, Emilia Zari, Ahmad Zareei, Matheus C. Fernandes, Laura Sumner, Jonathan Alvarez, and Katia Bertoldi. Inverse design of inflatable soft membranes through machine learning. *Advanced Functional Materials*, page 2111610, 2022.

[29] 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, 2021.

[30] Justin Fu and Sergey Levine. Offline model-based optimization via normalized maximum likelihood estimation. *CoRR*, abs/2102.07970, 2021. URL `https://arxiv.org/abs/2102.07970`.

[31] Robert Geirhos, Carlos R Medina Temme, Jonas Rauber, Heiko H Schütt, Matthias Bethge, and Felix A Wichmann. Generalisation in humans and deep neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7549–7561, 2018.

[32] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[33] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1487–1495, 2017.

[34] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, JoséMiguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 02 2018.

[35] Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *Nature Communications*, 12(1):5721, Oct 2021.

[36] Jessica B Hamrick, Abram L Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Buesing, Petar Veličković, and Théophane Weber. On the role of planning in model-based deep reinforcement learning. *arXiv preprint arXiv:2011.04021*, 2020.

[37] Paul Z. Hanakata, Ekin D. Cubuk, David K. Campbell, and Harold S. Park. Forward and inverse design of kirigami via supervised autoencoder. *Phys. Rev. Research*, 2:042006, Oct 2020.

[38] Nikolaus Hansen, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea. *The CMA Evolution Strategy: A Comparing Review*, pages 75–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[39] 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.

[40] Ping He, Charles A Mader, Joaquim RRA Martins, and Kevin J Maki. Dafoam: An open-source adjoint framework for multidisciplinary design optimization with openfoam. *AIAA journal*, 58 (3):1304–1319, 2020.

[41] Matteo Hessel, David Budden, Fabio Viola, Mihaela Rosca, Eren Sezener, and Tom Hennigan. Optax: composable gradient transformation and optimisation, in JAX!, 2020.

[42] Stephan Hoyer, Jascha Sohl-Dickstein, and Sam Greydanus. Neural reparameterization improves structural optimization. *CoRR*, abs/1909.04240, 2019.

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

[44] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[45] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

[46] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[47] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *CoRR*, abs/1912.13465, 2019. URL `http://arxiv.org/abs/1912.13465`.

[48] Siddhant Kumar, Stephanie Tan, Li Zheng, and Dennis M. Kochmann. Inverse-designed spinodoid metamaterials. *Nature Computational Materials*, 6(1):73, 2020.

[49] Charles L Ladson. *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.

[50] Jichao Li, Mengqi Zhang, Joaquim Martins, and Chang Shu. Efficient aerodynamic shape optimization with deep-learning-based geometric filtering. *AIAA Journal*, 58:1–17, 07 2020.

[51] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2019.

[52] Zhaocheng Liu, Dayu Zhu, Sean P. Rodrigues, Kyu-Tae Lee, and Wenshan Cai. Generative model for the inverse design of metasurfaces. *Nano Letters*, 18(10):6570–6576, Sep 2018. ISSN 1530-6992.

[53] Michael Lutter, Leonard Hasenclever, Arunkumar Byravan, Gabriel Dulac-Arnold, Piotr Trochim, Nicolas Heess, Josh Merel, and Yuval Tassa. Learning dynamics models for model predictive agents. *arXiv preprint arXiv:2109.14311*, 2021.

[54] Sean Molesky, Zin Lin, Alexander Y Piggott, Weiliang Jin, Jelena Vucković, and Alejandro W Rodriguez. Inverse design in nanophotonics. *Nature Photonics*, 12(11):659–670, 2018.

[55] OpenCFD Ltd. OpenFOAM CFD solver. `www.openfoam.org`, 2021.

[56] François Osiurak and Arnaud Badets. Tool use and affordance: Manipulation-based versus reasoning-based approaches. *Psychological review*, 123(5):534, 2016.

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

[58] Janeita Reid. Free-form deformation. `dafoam.github.io/docs/FFD/main.pdf`, 2021.

[59] Simiao Ren, Willie Padilla, and Jordan Malof. Benchmarking deep inverse models over time, and the neural-adjoint method. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 38–48. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/007ff380ee5ac49ffc34442f5c2a2b86-Paper.pdf`.

[60] Li Rhie, C Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21(11):1525–1532, 1983.

[61] Reuven Y Rubinstein and Dirk P Kroese. The cross-entropy method: A unified approach to monte carlo simulation, randomized optimization and machine learning. *Information Science & Statistics, Springer Verlag, NY*, 2004.

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

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

[64] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. *Conference on Robot Learning (CoRL)*, 2018.

[65] Ney Secco, Gaetan K. W. Kenway, Ping He, Charles A. Mader, and Joaquim R. R. A. Martins. Efficient mesh generation and deformation for aerodynamic shape optimization. *AIAA Journal*, 2021.

[66] Wuxin Sha, Yan Li, Shun Tang, Jie Tian, Yuming Zhao, Yaqing Guo, Weixin Zhang, Xinfang Zhang, Songfeng Lu, Yuan-Cheng Cao, and Shijie Cheng. Machine Learning in polymer informatics. *InfoMat*, 3(4):353–361, 2021.

[67] Robert W Shumaker, Kristina R Walkup, and Benjamin B Beck. *Animal tool behavior: the use and manufacture of tools by animals*. JHU Press, 2011.

[68] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf`.

[69] Erik Talvitie. Model regularization for stable sample rollouts. In *UAI*, pages 780–789, 2014.

[70] Marc A Toussaint, Kelsey Rebecca Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Proceedings of the Robotics: Science and Systems, RSS 2018*, 2018.

[71] Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Conservative objective models for effective offline model-based optimization. In *International Conference on Machine Learning*, pages 10358–10368. PMLR, 2021.

[72] Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. Improving multi-step prediction of learned time series models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[73] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

[74] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

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

[76] Sunwoong Yang, Sanga Lee, and Kwanjung Yee. Inverse design optimization framework via a two-step deep learning approach: application to a wind turbine airfoil. *Engineering with Computers*, 2022. doi: 10.1007/s00366-022-01617-6. URL `https://doi.org/10.1007/s00366-022-01617-6`.

[77] Sihyun Yu, Sungsoo Ahn, Le Song, and Jinwoo Shin. Roma: Robust model adaptation for offline model-based optimization. *Advances in Neural Information Processing Systems*, 34, 2021.

[78] Li Zheng, Siddhant Kumar, and Dennis M. Kochmann. Data-driven topology optimization of spinodoid metamaterials with seamlessly tunable anisotropy. *Computer Methods in Applied Mechanics and Engineering*, 383:113894, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See e.g. failure cases in Section D.3.1 and Discussion.
   (c) Did you discuss any potential negative societal impacts of your work? [No] We didn't identify anything specific to this paper, the same considerations as for most work in ML apply (e.g. cost/impact of compute).
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The core model we use (MeshGraphNets) has an open-source implementation available, as do the solvers we used to for the task domains (cited in the paper). Detailed description of the task setup can be found in Appendix C.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix A, Appendix B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] We didn't calculate the total amount; but model training was done on typical hardware (up to 4 TPUv3/v100 GPUs, see Appendix B).

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We didn't use any data, but cite the model, and solvers used for the task domains.
   (b) Did you mention the license of the assets? [No] All of them are available under open-source licenses.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

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 $b_2 = 0.999$ [41].

	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

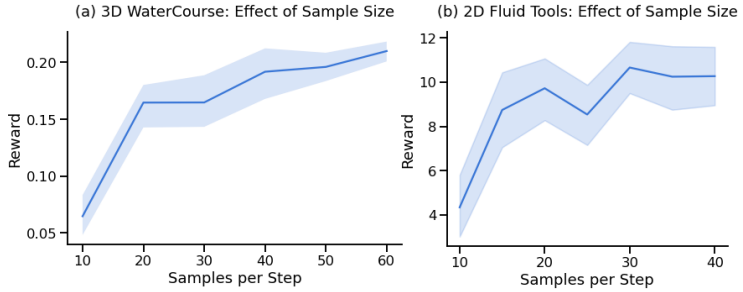

Figure A.1: For CEM, increasing population size, while more computationally expensive, can lead to improvements in performance. (a) In the 3D WATERCOURSE domain, CEM benefits from large sample sizes, although returns are diminishing for sizes beyond 40 (*Direction* task, 36 design parameters). (b) In 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 [63] and MESHGRAPHNETS [57]: 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 [11]. 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. The models were trained on 4 TPUv3 for 4-10 days.

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 [18] 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 $[x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}}]$. Certain task parameters were varied for ablation experiments in Figure 4 (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.3.

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 [43]. 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

	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 $[x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}}]$.

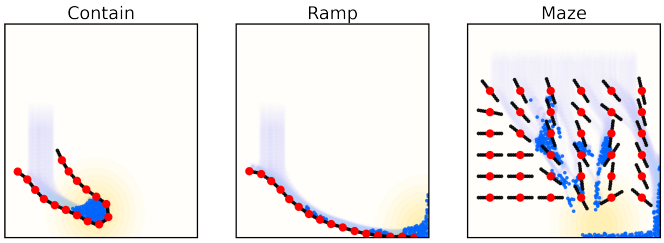

Figure A.2: Visualization of the design space parameterization for the 2D FLUID TOOLS task. Each red dot corresponds to the anchor points (*Contain* and *Ramp*) and center of rotation (*Maze*) being optimized.

calculated as

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

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 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. [63]. 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.

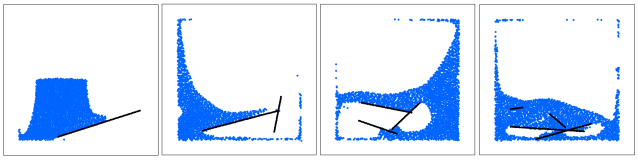

Figure A.3: Four examples of trajectories from the WATERRAMPS dataset released by [63] used as training data for the supervised prediction model.

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 $\mathbf{c} = (0.3, 0.5, 0.5)$ in the simulation domain. While we could directly mapping the parameters to coordinates, we use the design function $y_i = \gamma_H \tanh(\phi_{\text{map}_i})$ ($\gamma_H = 0.3$ 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

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

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 $f_{R_{\text{dir}}} - f^{\text{initial}}_{R_{\text{dir}}}$, 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 $180\,\text{deg}$. 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.

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

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 [8]. 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 5a). 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 [58], and the mesh is deformed using IDWarp [65]. 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. [49]. 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 [55]. 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.

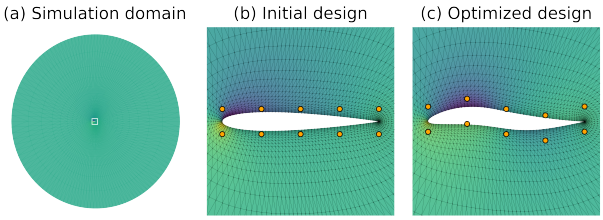

Figure A.4: (a) Aerodynamics are computed on a large 4158 node mesh centered around the airfoil, with the closeup regions around the airfoil in (b, c) marked as a white square in the center. (b, c) Pressure predictions and control points (orange) for the initial and final optimized wing design (Ensemble-5 model).

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 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.10), 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.

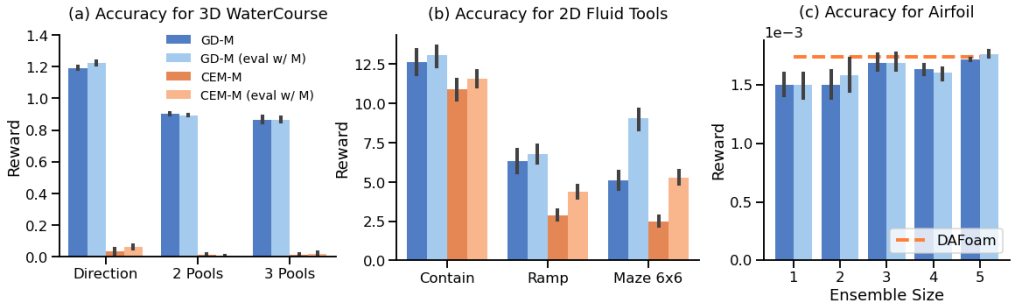

Figure A.5: (a) For the 3D WATERCOURSE domain, reward predicted by the learned model (GD-M eval w/ M, CEM-M eval w/ M) is very close to the ground-truth simulator evaluation (GD-M, CEM-M) for all tasks. (b) This is also true for the 2D FLUID TOOLS domain, though the reward is slightly overestimated when using the model. This effect is amplified in *Maze*, where the ground-truth dynamics sometimes struggles to correctly simulate "sticky" bottlenecks (see Figure A.10). (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 Other optimizers

In our experiments in Section 5, we chose the cross-entropy method (CEM) as a popular optimizer representative of model predictive control methods [26, 20], but there are several other possible choices of optimizer. In particular, here we provide results from optimizers that represent two

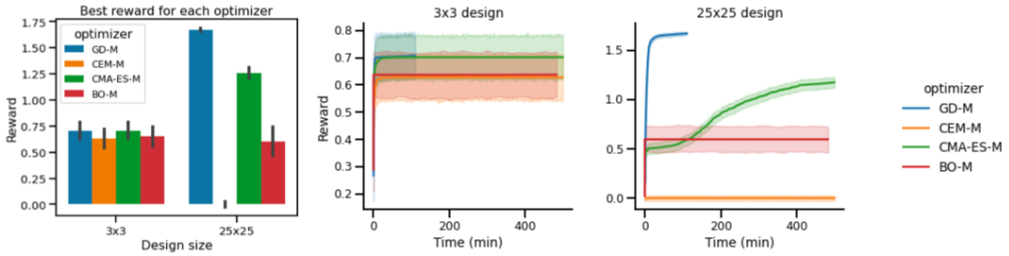

Figure A.6: Performance of alternative optimizers as a function of optimization time. Left: overall performance for each optimizer on the 3D Directions task. Center/Right: performance with a design space consisting of a 3x3 and 25x25 grid, respectively. For large design spaces, GD-M converges significantly faster than alternative optimization schemes, and leads to higher rewards. Error bars and shaded regions represent 95% confidence intervals over 9 directions and 10 seeds per direction.

alternative classes: bayesian optimization [68] (BO, which uses active sampling to explore the design space more effectively), and CMA-ES [38] (which is an evolutionary search method and represents a more sophisticated sampling technique than CEM).

We used the Bayesian Optimization implementation from [33] with their automatic hyperparameter tuning that uses automatic relevance determination (ADR) [74] with a Matérn kernel and the expected improvement (EI) acquisition function. For CMA-ES we swept across the top_k to keep, the smoothing parameter across iterations, the number of samples to use in each step, and the initial standard deviation of the sampling distribution. These were fit on the direction template with a single direction across 2 seeds per hyperparameter. The best performing hyperparameters for CMA-ES were to keep the top 10% of samples, no smoothing across optimization iterations, 40 samples per optimization step, and an initial standard deviation of 0.025. We show results for the 3D WATERCOURSE Direction template to showcase the scalability of both methods at the 3x3 design space and 25x25 design space resolutions.

For a low-dimensional design space (3x3), all optimizers perform comparably. While CMA-ES and GD slightly outperform CEM and BO, the differences are minor. For larger design space sizes (25x25), GD-M significantly outperforms all alternative optimizers. Given enough optimization time, CMA-ES is able to achieve about three quarters of the reward obtained by GD, although it takes roughly 50× as long to do so, as, like CEM, it needs to sample a significantly larger number of environment calls for each optimization step. It outperforms CEM because it represents a full covariance matrix across the top samples in the population, rather than a single scalar.

D.3 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.7). 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.

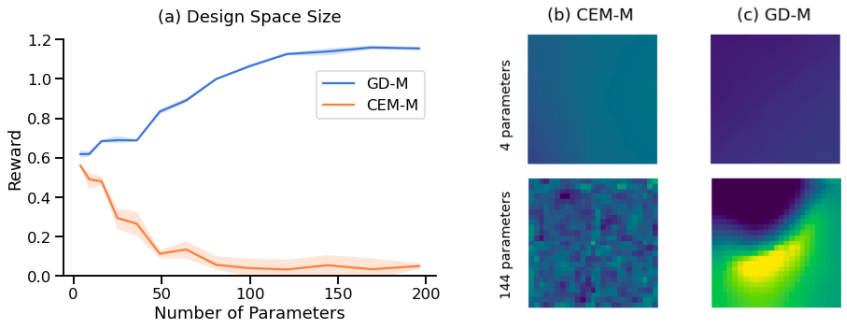

Figure A.7: Performance on the 3D WATERCOURSE *Direction* task with variable design space resolution: GD performs well for large design spaces, while CEM performance quickly drops with increased number of design parameters. (b) Examples of CEM designs at two different design space resolutions. (C) Examples 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.8). 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.8 top), the overall reward achieved is similar.

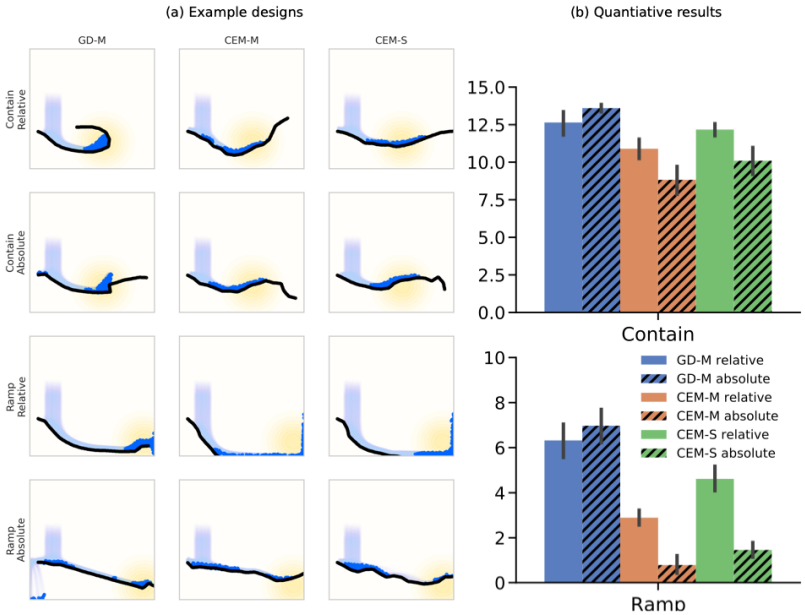

Figure A.8: **(a)** Example solutions for each optimizer across the *Contain* and *Ramp* tasks when optimizing over *Relative* vs *Absolute* angles. **(b)** Mean reward with 95% confidence intervals obtained by each optimizer across the *Contain* and *Ramp* tasks when optimizing over *Relative* vs *Absolute* angles.

D.3.1 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.9). 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 [70].

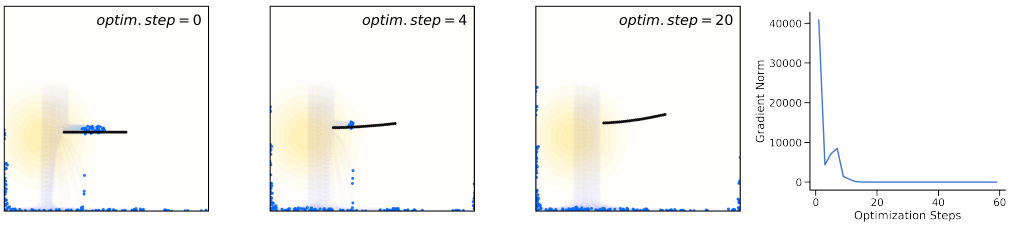

Figure A.9: Failure mode of the GD optimizer: in some instances where the translation of the tool is included in the design, the tool may end up outside of the scope of the fluid. In this cases, the optimization can no longer recover as it will get zero gradients from there on.

D.4 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 [43] 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.10).

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.

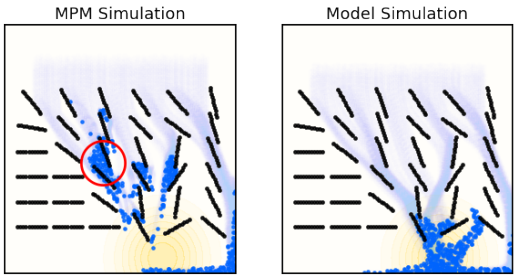

Figure A.10: Left: MPM simulation [43] of a problem with many separate solid objects. As highlighted in the red circle, the MPM solver struggles with water movement between obstacles, often creating artificially sticky bottlenecks. Right: Learned model rollout for the same setup. The model rollout looks significantly more plausible, without any "stickiness" artifacts. Please see `https://sites.google.com/view/optimizing-designs` for videos demonstrating this effect clearly.

D.4.1 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.

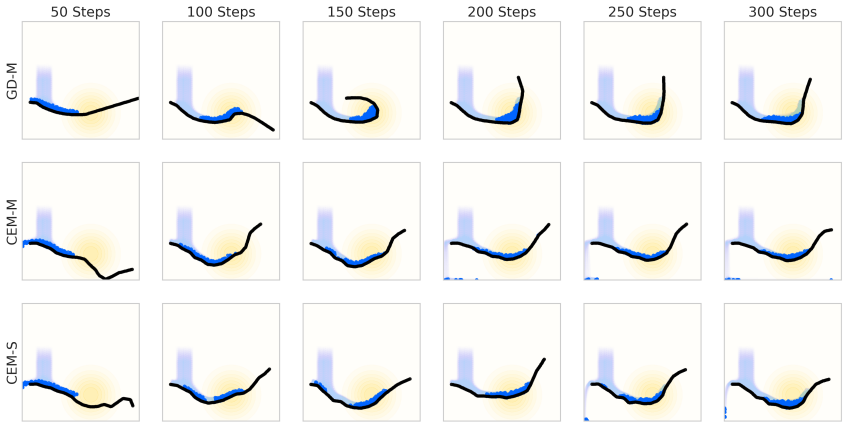

Figure A.11: Example solutions for each optimizer across the range of rollout lengths sampled for *Contain* in Figure 4.

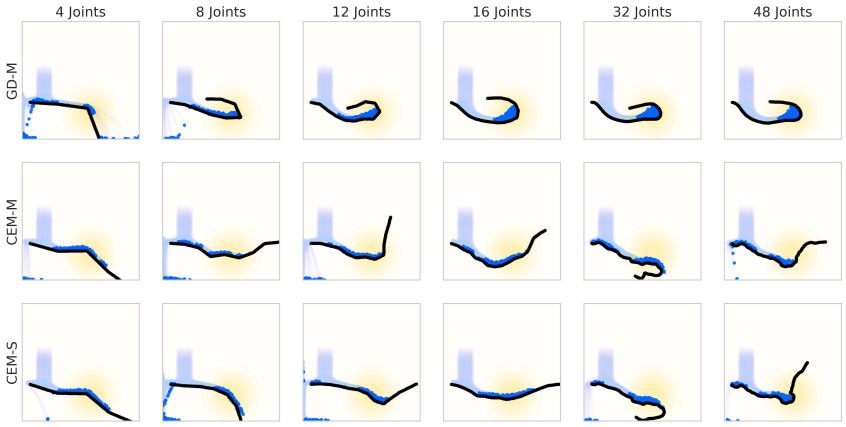

Figure A.12: Example solutions for each optimizer across the range of joint angle numbers sampled for *Contain* in Figure 4.

Figure A.13: Example solutions for each optimizer across the range of grid sizes sampled for *Maze* in Figure 4.

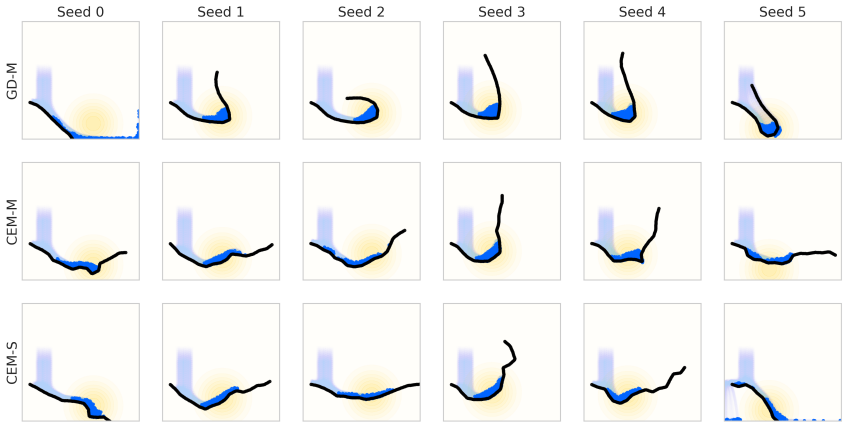

Figure A.14: Example solutions on *Contain* task for each optimizer across 6 random seeds.

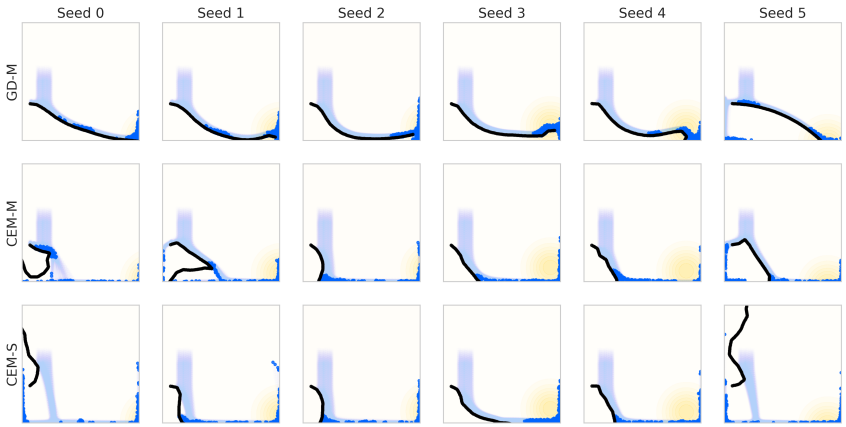

Figure A.15: Example solutions on *Ramp* task for each optimizer across 6 random seeds.

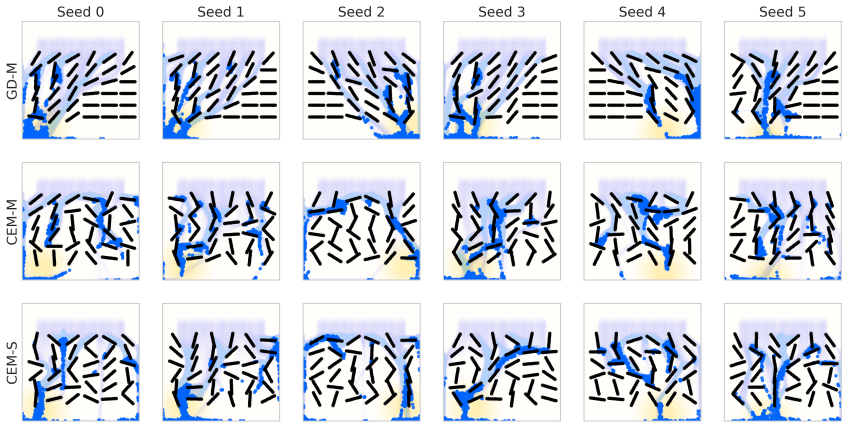

Figure A.16: Example solutions on *Maze* for each optimizer across 6 random seeds.

D.5 End2End Surrogate Reward Models

D.5.1 3D Watercourse

In this section we trained an end-to-end surrogate to map design parameters (ϕ_{map}) directly to rewards on the same designs as those contained in the training set for the GNN model (Section C.2), but extended with the rewards predicted by the ground truth simulator in each of the $\mathbf{d} = 8$ target angles of the *Direction* task, so that $\mathcal{D}_{sur} = \{(\phi_{\text{map}}, f_{R_{\text{dir}}}(\phi_{\text{map}}, \mathbf{d}))^{(i)}\}_{i=0}^{1000}$. We used a ResNet-18 as the reward predictive model with a 25×25 dimensional input corresponding to the heightfield mesh, and we trained a separate model for each target direction.

The model could fit the training set, as well as an in-distribution test set very well (MSE=0.0070 ± 0.0025). However, the model performs predictably poorly out of distribution (Figure A.17): It (1) severely underestimates the reward of high-scoring designs found using GD-M (predicted=0.33 ± 0.06,

actual=1.21 ± 0.18, MSE=0.813 ± 0.344), and (2) obtains poor designs when using this model in an GD optimizer (predicted=1.12 ± 0.31, actual=0.40 ± 0.05, MSE=0.615 ± 0.403).

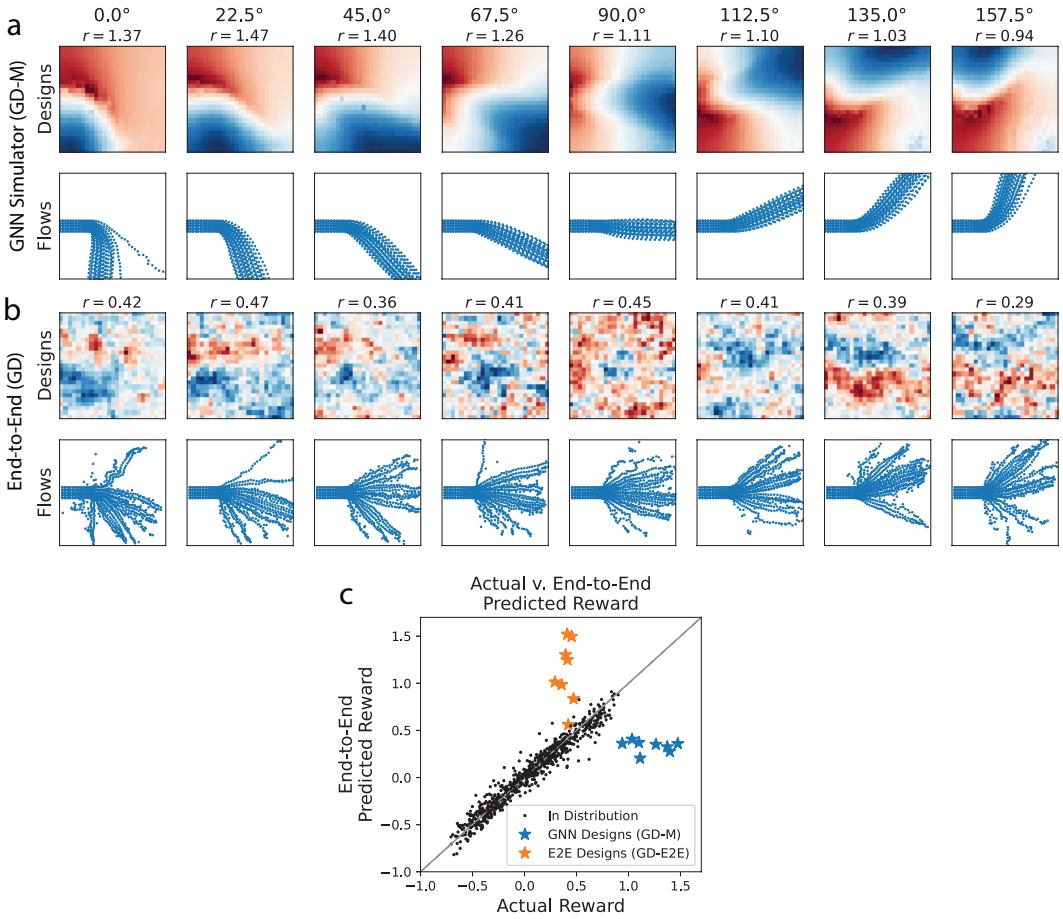

Figure A.17: Comparison of the end-to-end surrogate model (GD-E2E), which consists of a ResNet-18 [39] trained to predict reward directly from 25×25 grid of design parameters, on the 3D WATERCOURSE *Direction* task. (a - top row) an example of a design solution optimized with the Learned GNN Simulator (GD-M) for each direction. (a - bottom row) top view of the fluid flow for each solution using the design pictured above. (b - top row) an example of a design solution optimized using the trained End-to-End surrogate model (GD-E2E) for each direction. (b - bottom row) top view of the fluid flow for each solution using the design pictured above. (c) Predicted vs Ground Truth (evaluated using f_s) reward for different sets of designs: (black dots) held-out test data sampled from the same distribution used to sample training data, (blue stars) optimized designs shown in (a) found using the Learned GNN Simulator (GD-M), (orange stars) optimized designs shown in (b) found using the End-to-End model (GD-E2E).

D.5.2 2D FLUID TOOLS

Unlike the dataset used in the previous section, the WATERRAMPS dataset does not share a design space with the 2D FLUID TOOLS tasks (i.e. there are no joints, only a small number of line segments), so we were unable to train a reward model end-to-end as above on this data. Therefore, we instead computed the reward for 1000 random designs (the same size as the training set for the GNN model) for the 2D FLUID TOOLS-*Contain* task using 4 up to 48 design parameters (Table A.2) to assess the difficulty of creating a dataset which includes high rewards using random sampling.

The maximum reward found decreased quickly for > 12 design parameters: i.e., for 48 joints, the highest reward encountered is $3\times$ less than the average reward design obtained using the GD-M

Number of joints	4	8	12	16	20	24	28	32	36	40	44	48
Random design reward	12.5	11.4	9.8	8.6	8.0	6.9	5.8	5.5	5.2	4.7	4.8	4.4

Table A.2: Maximum reward found via random exploration of 1k designs across different joint dimensions on the 2D FLUID TOOLS-*Contain* task.

(Figure 4-b) for the same dimensionality. Hence, a reward model trained on such data would not perform very well, as effective designs are highly out-of-distribution.

D.6 Further tasks for 3D WATERCOURSE

To demonstrate the flexibility and further scaleability of the approach, we designed two further tasks for the 3D WATERCOURSE environment that require optimizing **two** meshes in order to manipulate the flow of a fluid: DOUBLEMESHREROUTE and DOUBLEMESHRAMP. Unlike in the original 3D WATERCOURSE tasks, in these two tasks the global rotation θ_g (in the z plane) of the meshes can be optimized as well as the heightfields, leading to a design dimensionality of 1252 (each mesh has 626 parameters to optimize). Both tasks are particularly out of distribution for this environment, since there was only ever one mesh (with a sinewave heightfield) without any vertical rotation seen during training.

In both cases, the positions of the meshes are fixed to $(-0.25, -0.1, 0.0)$ and $(0.7, -0.2, 0.0)$ respectively, allowing them to be offset in both the height and distance from the water emitter.

D.6.1 DOUBLEMESHREROUTE task

In this task, the reward region is placed at $(0.5, 0.0, 0.5)$, which is just underneath the edge of the first mesh. In order to move the fluid to this location, the fluid needs to be guided by the first mesh into a stream, and then rerouted by the second mesh to bounce back and underneath the first mesh. We define the reward as the Gaussian probability for each particle $\mathbf{u_v}$under $\hat{\mu}$ with $\mu = (0.5, 0.0, 0.5)$ (a region at (0.5, 0.5) on the floor), i.e.

$$f_{R_{\text{reroute}}} := \text{mean}_v(\mathcal{N}(\mathbf{u_v}; \hat{\mu}, \sigma)) - \gamma_R \,\text{mean}(\nabla \phi_{\text{map}})$$

with $\sigma = 0.1$ and a regularization term γ_R = 100 to provide smoothness, as in the original 3D WATERCOURSE tasks.

The mesh rotations are initialized to $20°$ and $-40°$ respectively, and then allowed to be optimized further along with the heightfields (rotations end up at $35.6°$ and $-44.2°$ respectively after optimization). After 100 optimization steps, in this task the designs achieve a reward of 1.16 when evaluated with the ground truth simulator.

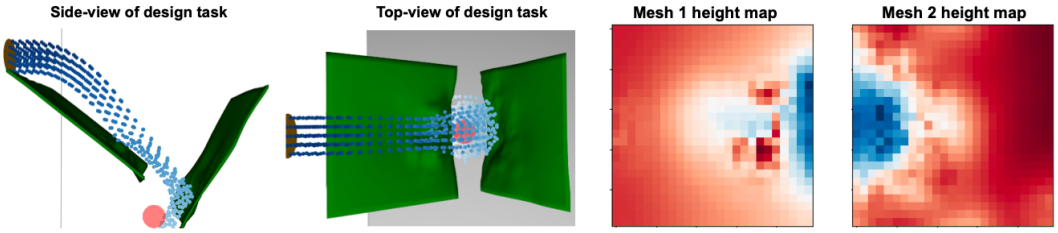

Figure A.18: Double mesh task where both the global rotation and the heightfield of **two meshes** can be optimized, a total of 625×2 + 2 = 1252 parameters. The model is the same one as used for the Direction and Cluster templates. This design task involves rerouting the fluid to the red ball region. To do so, the meshes must cooperate to reroute the fluid against its direction of travel using steep rotations. The mesh was never rotated in the training data, and only one mesh was ever seen in the training data.

D.6.2 DOUBLEMESHRAMP task

In this task, the reward region is placed at $(2.5, 0.0, 0.5)$, which is past the edge of the second mesh. In order to move the fluid to this location, the fluid needs to be guided by the first mesh into a stream, and then rerouted by the second mesh to bounce back and underneath the first mesh. We define the reward as the Gaussian probability under $\hat{\mu}$ with $\mu = (2.5, 0.0, 0.5)$, i.e.

$$f_{R_{\text{ramp}}} := \text{mean}_v(\mathcal{N}(\mathbf{u_v}; \hat{\mu}, \sigma)) - \gamma_R \,\text{mean}(\nabla \phi_{\text{map}})$$

with $\sigma = 0.1$ and a regularization term γ_R = 100.

The mesh rotations are initialized to $0°$, and then allowed to be optimized further along with the heightfields (rotations end up at $-4.4°$ and $-7.8°$ respectively after optimization). After 100 optimization steps, in this task the designs achieve a reward of 0.52 when evaluated with the ground truth simulator.

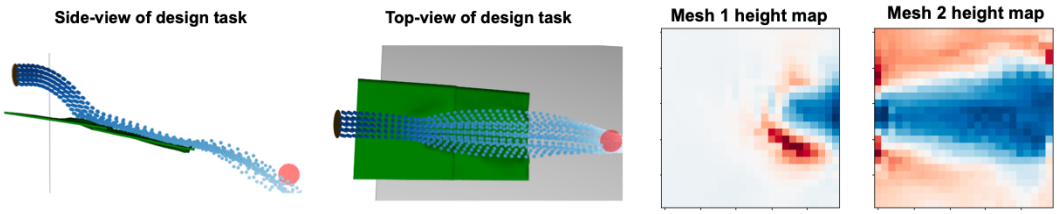

Figure A.19: Double mesh task where both the global rotation and the heightfield of **two meshes** can be optimized, a total of 625×2 + 2 = 1252 parameters. The model is the same one as used for the Direction and Cluster tasks. This design task involves rerouting the fluid to the red ball region. To do so, the meshes must cooperate to extend a ramp so that the water makes it all the way to the far side. Only a single, non-rotated mesh was seen in the training data.