Amortized Synthesis of Constrained Configurations Using a Differentiable Surrogate

Xingyuan Sun[1], **Tianju Xue**[2], **Szymon Rusinkiewicz**[1], **Ryan P. Adams**[1]
[1]Department of Computer Science [2]Department of Civil and Environmental Engineering
Princeton University
{xs5, txue, smr, rpa}@princeton.edu

Abstract

In design, fabrication, and control problems, we are often faced with the task of *synthesis*, in which we must generate an object or configuration that satisfies a set of constraints while maximizing one or more objective functions. The synthesis problem is typically characterized by a physical process in which many different realizations may achieve the goal. This many-to-one map presents challenges to the supervised learning of feed-forward synthesis, as the set of viable designs may have a complex structure. In addition, the non-differentiable nature of many physical simulations prevents efficient direct optimization. We address both of these problems with a two-stage neural network architecture that we may consider to be an autoencoder. We first learn the decoder: a differentiable surrogate that approximates the many-to-one physical realization process. We then learn the encoder, which maps from goal to design, while using the fixed decoder to evaluate the quality of the realization. We evaluate the approach on two case studies: extruder path planning in additive manufacturing and constrained soft robot inverse kinematics. We compare our approach to direct optimization of the design using the learned surrogate, and to supervised learning of the synthesis problem. We find that our approach produces higher quality solutions than supervised learning, while being competitive in quality with direct optimization, at a greatly reduced computational cost.

1 Introduction

One of the ambitions of artificial intelligence is to automate problems in design, fabrication, and control that demand efficient and accurate interfaces between machine learning algorithms and physical systems. Whether it is optimizing the topology of a mechanical structure or identifying the feasible paths for a manufacturing robot, we can often view these problems through the lens of *synthesis*. In a synthesis task, we seek configurations of a physical system that achieve certain desiderata while satisfying given constraints; *i.e.*, we must optimize a physically-realizable design.

In this work, *design* refers to the parametric space over which we have control and in which, *e.g.*, we optimize. A *realization* is the object that arises when the design is instantiated, while *goal* refers to its desired properties. For example, in fabrication, the design might be a set of assembly steps, the realization would be the resulting object, while the goal could be to match target dimensions while maximizing strength. Synthesis, then, refers to finding a design whose realization achieves the goal.

Synthesis problems are challenging for several reasons. The physical realization process may be costly and time-consuming, making evaluation of many designs difficult. Moreover, the realization process—or even a simulation of it—is generally not differentiable, rendering efficient gradient-based methods inapplicable. Finally, there may be a many-to-one map from the parametric space of feasible and equally-desirable designs to realizations; *i.e.*, there may be multiple ways to achieve the goal.

Surrogate modeling is widely used to address the first two challenges, though it can still be expensive because of the need for optimization, sampling, or search algorithms to find a feasible design. More seriously, the third challenge—lack of uniqueness—creates difficulties for naïve supervised learning

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

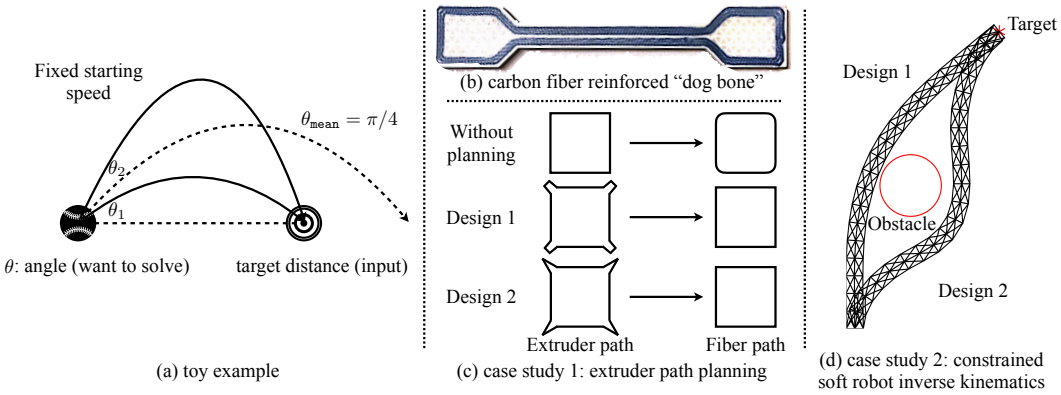


Figure 1: (a) For a fixed and large-enough starting speed, there exist exactly two angles such that the ball will hit the target, where the mean of these two angles is $\pi/4$. (b) Some 3D printers utilize fibers to reinforce the thermoplastic print. (c) For such printers, fiber is laid out along an extruder path but deforms into a smoothed version due to the fiber's high stiffness and low stretch. Our goal is to generate extruder paths that compensate for the smoothing, but multiple extruder paths can result in the same target shape, such as a square. (d) In soft robot inverse kinematics, we control the stretch ratios of both the left- and right-hand sides of a snake-like robot. Our goal is to reach the target while avoiding an obstacle but, as is illustrated, the solution is not unique – two different designs are shown.

approaches to synthesis. Specifically, consider generating many design/realization pairs, evaluating the constraints and objectives on the realizations, and attempting to learn a supervised map from goal back to design. When multiple designs lead to the same realization, or multiple realizations achieve the same goal, the supervised learner is penalized for producing designs that are valid but happen to not be the ones used to generate the data. Moreover, this approach may learn to produce an "average" design that is actually incorrect. Figure 1a shows a cartoon example: if the goal is to throw a ball to reach some target distance, there are two possible launch angles (designs) resulting in landing points (realizations) at the correct spot. Performing least-squares regression from distance to angle on the full set of distance/angle pairs, however, learns an average angle that does not satisfy the goal.

To address these challenges, we propose to use a two-stage neural network architecture that resembles an autoencoder. One stage (the decoder) acts as a differentiable surrogate capturing the many-to-one physical realization process. The other stage (the encoder) maps from a goal back to a design but, critically, it is trained end-to-end with a loss in the space of realizations that flows back through the decoder. Thus the encoder—our central object of interest for synthesis—is not constrained to match a *specific* design in a training dataset, but instead is tasked with finding *any* design that meets the desiderata of the realized output. The result is a neural network that performs *amortized* synthesis: it is trained once, and at run time produces a design that is approximately optimal, using only a feed-forward architecture. Note that our method is not an autoencoder, as the design is not a lower-dimensional representation of the goal, and the encoder and the decoder are trained in separate stages.

Our method places a number of requirements on the synthesis problem. First, to train the surrogate, we need data pairs of designs and realizations. Commonly, this would require us to generate designs, in which a substantial amount of designs are viable, and simulate them on a simulator. Second, given our current setting, the physical realization process needs to be deterministic. Third, to train the encoder, the synthesis problem should have a clear objective function, or at least we can quantify the objective. Finally, given our current feed-forward setting, we consider synthesis problems that need only one valid design, although we may further extend our method by using a generative encoder.

In this work, we demonstrate this two-stage approach on a pair of specific design tasks. The first case study is extruder path planning for a class of 3D printers (the Markforged Mark Two) that can reinforce polymer layers with discrete fibers (Figure 1b). Since the fibers are stiff, their shape is deformed after extrusion (Figure 1c, top row), and our task is to find an extruder path that results in a given fiber shape. As shown in Figure 1c, this problem has the many-to-one nature described above: for a small error tolerance on fiber path, there exist infinitely many extruder paths, which may even look very different. The second case study is constrained soft robot inverse kinematics. In this work, we use a simulation of a snake-like soft robot as in Xue et al. [97], in which we can control the stretch ratios of each individual segment on both sides of the robot. The robot has to reach

a target while avoiding an obstacle, and the locations of both are input goals. As before, there may be multiple solutions for a given goal (*i.e.*, locations of target and obstacle), as shown in Figure 1d.

For both case studies, we compare to two baseline algorithms. In `direct-learning`, a neural network for the synthesis problem (*i.e.*, from goal to design) is trained in a supervised manner on a set of designs. Since this effectively averages designs in the training dataset, as argued above, our method outperforms it significantly. The second baseline is `direct-optimization`, which uses a gradient-based method (BFGS) to optimize for each new design separately, given access to the trained differentiable surrogate for the realization process (decoder). Our method is competitive with this rough "performance upper bound" while using dramatically lower computational resources.

2 Related work

Machine learning applications in synthesis problems. In synthesis tasks, the aim is to find a design solution such that its realization achieves one or more given goals. Usually, the solution is non-unique, and only one is needed. In molecule discovery, one would like to find a molecule which has some desired properties (*e.g.*, minimal side effects, efficacy, metabolic stability, growth inhibition) [20, 46, 71, 41, 83]. See Vamathevan et al. [90], Chen et al. [16] for surveys on machine/deep learning in drug discovery. Challenges of molecule discovery include discrete design space [32, 76] and limited data [47] due to difficulty in simulation. In materials synthesis, researchers seek materials with specific properties. See Bhuvaneswari et al. [7] for a review. Similarly, to obtain enough training data, researchers have put efforts into parsing and learning from scientific literature in natural language [52, 45]. In 3D shape generation, one wants to find a 3D shape that has some desired properties: properties like 2.5D sketches [95] that can be easily calculated and properties like functionality [34] that need to be human-labeled. In topology optimization, researchers aim to maximize the system's performance by optimizing the material layout given boundary conditions, constraints, external loads, etc [80, 5, 93]. Machine learning has been used to infer properties [75], find representations of designs [48, 99], and directly generate designs [55]. In program synthesis, the goal is to find programs that realize given intentions (*e.g.*, generating 3D shapes, answering visual questions) [35, 88, 98], where machine learning has been used to generate programs and execute programs. In this work, we propose an approach to learn a feed-forward neural network that can directly and efficiently produce a feasible solution to a synthesis problem that satisfies the requirements mentioned above.

Surrogate/oracle-based synthesis. Surrogate/oracle-based synthesis uses an auxiliary model—a surrogate (or sometimes called oracle)—that can evaluate qualities of a design without time-consuming laboratory experiments, while still being reasonably accurate [58, 9]. Surrogate models can be physics-based or approximation-based [57] (*i.e.*, empirical), and there are different modeling techniques for approximation-based surrogates, including polynomial regression, radial basis functions, Gaussian processes, and neural networks [58, 39]. Bhosekar and Ierapetritou [6] provide a review of surrogate-based methods, and see Koziel and Leifsson [56] for a general review of surrogate models in engineering. There are two major approaches: optimization and sampling. The most common approach is surrogate-based optimization; see Forrester and Keane [29] for a survey. Some application examples include: optimizing the parameters of a CPU simulator [72]; solving partial differential equations in service of PDE-constrained optimization [97]; and optimizing stochastic non-differentiable simulators [79]. Researchers have also developed methods to deal with the challenge that inputs can be out-of-distribution for the oracle [27, 30, 89]. On the other hand, sampling-based methods have several advantages: the design space can be discrete and can generate multiple designs [9, 10, 24]. Researchers have successfully applied sampling methods to problems in chemistry [32] and biology [36, 51, 73]. In this work, we use surrogate-based optimization as one of our baseline algorithms, and our proposed method uses a surrogate during training to optimize for a feed-forward network for the design problem.

Differentiable surrogate of losses in machine learning. Since some loss functions in machine learning are not differentiable (*e.g.*, IoU for rotated bounding boxes, 0-1 loss in classification), researchers have proposed to learn surrogates for them. Grabocka et al. [33] provided a formulation of surrogate loss learning and compared several learning mechanisms on some commonly used non-differentiable loss functions. Liu et al. [62] proposed a general pipeline to learn surrogate losses. Bao et al. [2], Hanneke et al. [40] provided theoretical analyses of surrogates for 0-1 loss in classification. Patel et al. [70], Nagendar et al. [64], Yuan et al. [100] explored the use of surrogate losses in various real-world tasks, including medical image classification, semantic segmentation,

and text detection and recognition. In this work, due to the non-uniqueness of design solutions, we further extend this idea from loss functions to more complex, physical realization processes.

Robot motion planning. Robot motion planning [59] can also be viewed as a form of synthesis, and there are different types of approaches to it. One popular strategy is *optimization*. For example, researchers have used evolutionary algorithms (EA) [60, 12, 42], including variants of genetic algorithms (GA) [18, 17, 87, 11, 50, 53, 28] and covariance matrix adaptation evolution strategy (CMA-ES) [37]. Another example is constrained optimization [94, 85, 21, 23], which includes extensions like sequential quadratic programming (SQP) [15, 37] and the DIRECT algorithm [91]. Other examples of optimization applied to robot motion planning include reinforcement learning [81, 25] and teaching-learning-based optimization [82]. Another popular class of methods is *sampling*, including simulated annealing (SA) [4, 69, 101], probabilistic roadmaps (PRM) [49], rapidly exploring random trees (RRT) [14, 4, 13], *etc*. Finally, *search* methods, *e.g*., A* search [38], have been used to solve motion planning problems. These works usually model the robot's physics directly without using a surrogate model and solve the design using methods including optimization, sampling, and search, which can be time-consuming. Our method amortizes the cost of inference, and can be used to solve planning problems with more complex physics that cannot be directly modeled. In our first case study, we plan for a trajectory of the extruder, and our design space is not the speeds of the motors but rather the coordinates of points along the trajectory. In the second case study, rather than solving a dynamic planning problem, we solve a static planning task on a soft robot, with the stretch ratios of all the controllable segments of the robot as the design space.

Path planning in 3D printing. Path planning is one of the most important problems in 3D printing, and people plan for different objectives: minimizing printing time, avoiding collision, faster planning, etc. Shembekar et al. [78] proposed a planning algorithm to build complex shapes with multiple curvatures and can avoid collisions. Ganganath et al. [31] minimized the printing time by modeling the task as a traveling salesman problem. Xiao et al. [96] speeded up path planning algorithms by introducing efficient topology reconstruction algorithms. Stragiotti [84] provided an optimization-based algorithm that minimizes compliance of a printed part. Asif [1] introduced a planning algorithm for continuous fiber and can generate a continuous deposition path. See Huang et al. [43] for a review of existing work in 3D printing path design. In this work, instead of the aforementioned objectives, we plan an extruder path to compensate for the deformation caused by the printing process. We demonstrate increased run-time efficiency by amortizing the cost of physical simulation of the printing process, learning a feed-forward network to output the extruder path.

3 Method

We formalize synthesis as a constrained optimization problem, denoting the set of allowed designs as Θ and the set of possible realizations as $\mathcal{U}$. There is a physical process that maps from design to realization that we denote as $U : \Theta \to \mathcal{U}$. Our goal may be a function of both the realization and the design, as designs may differ in, *e.g*., ease of manufacturing. Moreover, it may be appropriate to specify a parametric family of goals to accommodate related tasks, *e.g*., different target locations in inverse kinematics. The user expresses a ($\boldsymbol{g}$-indexed) family of design goals via a cost function denoted $\mathcal{L}_{\boldsymbol{g}} : \Theta \times \mathcal{U} \to \mathbb{R}$. The problem of interest is to optimize the cost function with respect to the design:

$$\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta}, \boldsymbol{u}) \quad \text{s.t.} \quad U(\boldsymbol{\theta}) = \boldsymbol{u}\,. \tag{1}$$

We can view this problem as a generalization of PDE-constrained optimization problems, where we have allowed for broader types of realizations than PDE solutions. Revisiting the challenges of synthesis problems articulated earlier, U may be expensive, non-differentiable, and non-injective, and $\mathcal{L}_{\boldsymbol{g}}$ may not a have a unique minimum.

Recalling the toy problem of Figure 1a: $\boldsymbol{\theta}$ is the angle at which the ball is thrown, $U(\boldsymbol{\theta}) = \boldsymbol{u}$ is where it lands, the goal $\boldsymbol{g}$ is a desired distance, *i.e*., a target realization, and a (design-independent) cost function might be the squared difference between the desired and actual realizations: $\mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta}, \boldsymbol{u}) = ||\boldsymbol{g} - \boldsymbol{u}||^2$.

Since the realization $\boldsymbol{u} = U(\boldsymbol{\theta})$ is unique for a specific design $\boldsymbol{\theta}$, we propose using a two-stage approach, which can be viewed as an autoencoder. We first learn a differentiable surrogate (decoder) $\hat{U}(\cdot)$ for the physical realization process from $\boldsymbol{\theta}$ to $\boldsymbol{u}$, and then learn an encoder $\phi(\cdot)$ from goal $\boldsymbol{g}$ to design $\boldsymbol{\theta}$, evaluating the design quality with the trained decoder. To build a dataset, we have to randomly sample designs and calculate realizations and goals from them, since the physical realization process $U(\cdot)$ is known, and the reverse direction (*i.e*., goal to design) is difficult as discussed before and is what we are trying to learn. Thus we first sample D designs $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_D$ and

build our dataset as $\mathcal{D} \coloneqq \big\{(\boldsymbol{\theta}_1, \boldsymbol{u}_1, \boldsymbol{g}_1), \cdots, (\boldsymbol{\theta}_D, \boldsymbol{u}_D, \boldsymbol{g}_D)\big\}$, where $\boldsymbol{u}_i \coloneqq U(\boldsymbol{\theta}_i)$ and $\boldsymbol{g}_i$ is the goal calculated from the realization $\boldsymbol{u}_i$. The calculation of the goal $\boldsymbol{g}_i$ summarizes properties that we care about in $\boldsymbol{u}_i$, and this process depends on the synthesis problem itself and how the cost function $\mathcal{L}_{\cdot}(\cdot, \cdot)$ is designed. We split $\mathcal{D}$ into $\mathcal{D}_{\texttt{train}}$, $\mathcal{D}_{\texttt{val}}$, and $\mathcal{D}_{\texttt{test}}$. We then train our surrogate $\hat{U}(\cdot)$ such that

$$\hat{U}^*(\cdot) \coloneqq \arg\min_{\hat{U}(\cdot)} \mathbb{E}_{(\boldsymbol{\theta}, \boldsymbol{u}, \cdot) \sim \mathcal{D}_{\texttt{train}}} \big[||\hat{U}(\boldsymbol{\theta}) - \boldsymbol{u}||^2\big], \tag{2}$$

so that $\hat{U}^*(\cdot)$ serves as a differentiable surrogate of the physical realization function $U(\cdot)$. We then use the trained decoder $\hat{U}^*(\cdot)$ to train an encoder $\phi(\cdot)$:

$$\phi^*(\cdot) \coloneqq \arg\min_{\phi(\cdot)} \mathbb{E}_{(\cdot, \cdot, \boldsymbol{g}) \sim \mathcal{D}_{\texttt{train}}} \big[\mathcal{L}_{\boldsymbol{g}}\big(\phi(\boldsymbol{g}), \hat{U}^*(\phi(\boldsymbol{g}))\big)\big]. \tag{3}$$

Note that although our method resembles an autoencoder, it is not an autoencoder, since we learn the decoder first and then the encoder rather than jointly. Besides, the design space is usually not a lower-dimensional representation of the goal space, and the cost function is usually not simply reconstruction loss.

4 Empirical evaluation approach

In this work, we use our method to solve two real problems as case studies: a task of optimizing the extruder path in additive manufacturing, and a task of actuating a soft robot in an inverse kinematics setting. While the details of the two differ, we evaluate them in the same way. In each case, we compare our algorithm with two baselines—`direct-learning` and `direct-optimization`—and evaluate our method in terms of relative quality and run time.

`direct-learning`: One natural baseline is to directly learn a model from the goal $\boldsymbol{g}$ to the design $\boldsymbol{\theta}$. Thus we introduce the `direct-learning` model $\phi_{\texttt{dl}}(\cdot)$:

$$\phi_{\texttt{dl}}(\cdot) \coloneqq \arg\min_{\phi(\cdot)} \mathbb{E}_{(\boldsymbol{\theta}, \cdot, \boldsymbol{g}) \sim \mathcal{D}_{\texttt{train}}} \big[||\boldsymbol{\theta} - \phi(\boldsymbol{g})||^2 + \mathcal{R}_{\texttt{dl}}(\phi(\boldsymbol{g}))\big]. \tag{4}$$

Note that, since there is no surrogate, we do not have access to the realization $\boldsymbol{u}$. We thus have to slightly adjust the cost function into a squared Euclidean distance part and a regularizer part $\mathcal{R}_{\texttt{dl}}(\cdot)$ (if there is one), the latter of which comes from the original cost function $\mathcal{L}_{\cdot}(\cdot, \cdot)$. We expect our method to perform much better than `direct-learning`.

`direct-optimization`: Since our surrogate is a neural network and therefore differentiable, another natural baseline is to directly optimize the design $\boldsymbol{\theta}$ with respect to the goal $\boldsymbol{g}$ by using a gradient-based optimizer (*e.g.*, BFGS), which provides us a rough performance "upper bound" on our method:

$$\phi_{\texttt{do}}(\boldsymbol{g}) \coloneqq \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta}, \hat{U}^*(\boldsymbol{\theta})). \tag{5}$$

Note that AmorFEA [97] is state-of-the-art method for PDE-constrained optimization, which uses the same approach as the `direct-optimization` baseline. We expect our method to have performance close to `direct-optimization`, while running orders of magnitude faster.

To evaluate our method and the baselines, we iterate over all $\boldsymbol{g}$ from $\mathcal{D}_{\texttt{test}}$, and evaluate the quality of each tuple $\big(\phi(\boldsymbol{g}), U(\phi(\boldsymbol{g})), \boldsymbol{g}\big)$, where we use the physical realization function $U(\cdot)$ rather than the learned surrogate $\hat{U}^*(\cdot)$. We run every experiment 3 times with random initialization of neural networks. More details are provided in each case study and in the appendix.

5 Case study: extruder path planning

3D printing (a.k.a. additive manufacturing) is the process of creating a 3D object from a 3D model by successively adding layers of material. It has a wide variety of applications in areas including aerospace, automotive, healthcare, and architecture [77]. Popular 3D printers use thermoplastic polymers (PLA, ABS, nylon, etc.) as the printing material, but these have limited strength. To address this issue, some recent printers support using strong fibers to reinforce the composite. In this work, we explore a 3D printer (the Markforged Mark Two) capable of extruding discrete fibers (fiberglass, kevlar, or carbon fiber) along a controllable path. However, since the fibers are stiff and non-stretchable, the printed fiber path will be "smoothed" compared to the extruder path. As shown in Figure 1c, without path planning, the fiber path will be severely deformed. We seek to find a general method that, given any desired fiber path, plans an extruder path to compensate for the deformation caused by the printing process. To the best of our knowledge, there is no existing automated method for this task, so it is worthwhile to tackle it using machine learning.

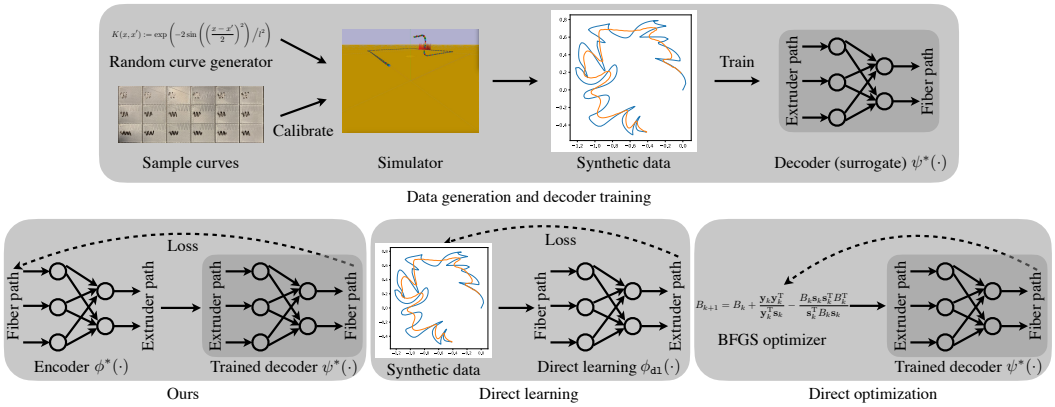


Figure 2: The pipeline for our method and two baselines for extruder path planning. We first generate data by building a simulator and calibrating it using a real printer, and we train the decoder and `direct-learning` using the synthetic dataset. We then train our method and build `direct-optimization` using the trained decoder.

5.1 Cost function

Following the notation in Section 3, we denote the target fiber path as $\boldsymbol{g} \in \mathbb{R}^{n\times 2}$: a path is represented as a series of n points, and n varies from path to path (at the scale of hundreds in our experiments). We denote the extruder path (the design) as $\boldsymbol{\theta} \in \mathbb{R}^{n\times 2}$ and its realization fiber path as $\boldsymbol{u} \in \mathbb{R}^{n\times 2}$. Our cost function $\mathcal{L}_{\cdot}(\cdot,\cdot)$ is defined as:

$$\mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta},\boldsymbol{u}) \coloneqq ||\boldsymbol{g}-\boldsymbol{u}||_2^2 + \lambda \cdot \mathcal{R}(\boldsymbol{\theta}), \tag{6}$$

where λ is a hyper-parameter and $\mathcal{R}(\cdot)$ is a smoothing regularizer that calculates the sum of squared empirical second-order derivatives of the extruder path:

$$\mathcal{R}(\boldsymbol{\theta}) \coloneqq \sum_{i=2}^{n-1} \left(\left(\frac{\boldsymbol{\theta}_{i+1}-\boldsymbol{\theta}_i}{||\boldsymbol{\theta}_{i+1}-\boldsymbol{\theta}_i||_2} - \frac{\boldsymbol{\theta}_i-\boldsymbol{\theta}_{i-1}}{||\boldsymbol{\theta}_i-\boldsymbol{\theta}_{i-1}||_2} \right) \Big/ \left(\frac{||\boldsymbol{\theta}_{i+1}-\boldsymbol{\theta}_i||_2 + ||\boldsymbol{\theta}_i-\boldsymbol{\theta}_{i-1}||_2}{2} \right) \right)^2, \tag{7}$$

where $\boldsymbol{\theta}_i \in \mathbb{R}^2$ is the i-th row of $\boldsymbol{\theta}$.

5.2 Evaluation metric

The most intuitive way to evaluate the quality of extruder path $\boldsymbol{\theta}$ is to measure the distance between $\boldsymbol{g}$, the desired fiber path, and $\boldsymbol{u}$, the fiber path we get by printing $\boldsymbol{\theta}$. Note that it is likely the model under-estimates or over-estimates the deformation of fiber introduced by printing, so both cases can happen when we print with path $\boldsymbol{\theta}$: we run out of fiber before we finish $\boldsymbol{\theta}$, or there is still some fiber remaining in the nozzle after we finish $\boldsymbol{\theta}$ (the total length of fiber is fixed given a desired fiber path $\boldsymbol{g}$). In other words, the $\boldsymbol{g}_i$'s and $\boldsymbol{u}_i$'s might not be synchronized. Thus, directly measuring the distance between $\boldsymbol{g}_i$ and $\boldsymbol{u}_i$ does not necessarily reflect the difference between desired fiber path and the fiber path we get. Therefore, to better measure the distance between $\boldsymbol{g}$ and $\boldsymbol{u}$ during testing, we use Chamfer distance [26], which was first proposed by Barrow et al. [3] as an image matching technique, later developed as a commonly used (semi)metric to measure the difference between two sets, and has been shown to have a higher correlation with human judgment compared to intersection over union and earth mover's distance [86]:

$$d_{\texttt{CD}}(\boldsymbol{g},\boldsymbol{u}) \coloneqq \frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n} \min_{j\in[1,n]} \|\boldsymbol{g}_i-\boldsymbol{u}_j\|_2 + \frac{1}{n}\sum_{j=1}^{n} \min_{i\in[1,n]} \|\boldsymbol{g}_i-\boldsymbol{u}_j\|_2 \right). \tag{8}$$

5.3 Implementation

The pipeline is shown in Figure 2. To generate the dataset for calibrating the decoder, we first use elliptical slice sampling [63] (New BSD License) to sample random extruder paths from a Gaussian process. We then use a physical simulator built using Bullet [22] (zlib License), calibrated to a real printer, to predict the realization (fiber path) for each extruder path. We generate 10,000 paths, split into 90% training, 5% validation, and 5% testing. For decoder, encoder, and `direct-learning`, we use an MLP with 5 hidden layers and ReLU as the activation function. The MLP takes 61 points as input and produces 1 point as output, and is applied in a "sliding window" fashion over the entire path (details in appendix). We train every model with a learning rate of 1×10^{-3} for 10 epochs using PyTorch [68] and Adam optimizer [54]. For `direct-optimization`, we use the BFGS implementation in SciPy [92]. More implementation details are included in the appendix.

Table 1: Path-planning evaluation of the average Chamfer distance on the test set evaluated in simulation

Regularizer weight	0.1	0.3	0.6	1.0	1.5
`direct-learning`	0.0314±0.0008	0.0319±0.0016	0.0502±0.0072	0.1007±0.0292	0.1457±0.0216
Ours	**0.0180±0.0004**	**0.0157±0.0003**	**0.0164±0.0004**	**0.0158±0.0002**	**0.0156±0.0002**
`direct-optimization`	0.0155±0.0002				

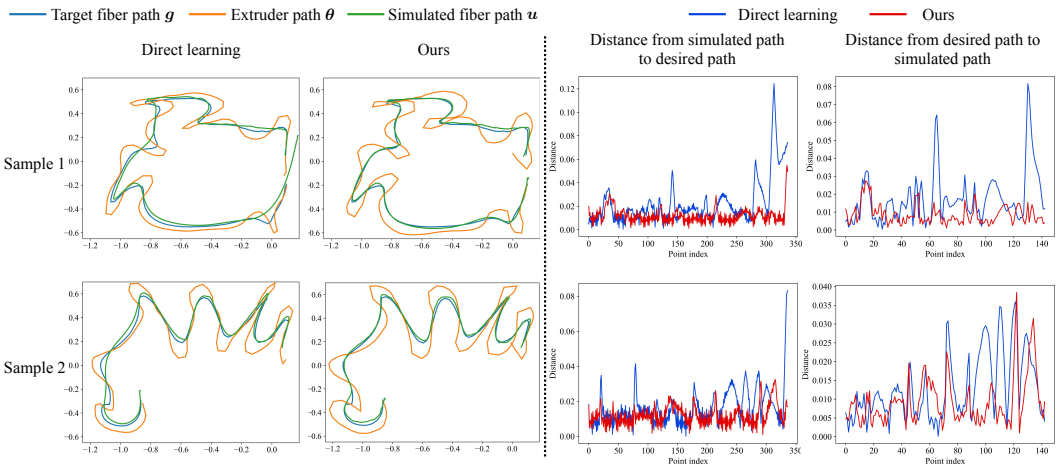


Figure 3: Path-planning evaluation of `direct-learning` *vs.* ours on the test set evaluated in simulation. We also visualize the Chamfer distance for each point on both the simulated fiber path and the desired fiber path.

5.4 Experiments

Fiber path quality evaluated in simulation. To quantitatively evaluate the effectiveness of our approach, the most straightforward way is to run its prediction on the simulator and see how close the simulated fiber path is to the input fiber path. We compare the performance of our approach (encoder), `direct-learning`, and `direct-optimization` on the test set of 500 paths, for different values of the regularization parameter λ. We report the average Chamfer distance (§ 5.2) with the standard error among 3 runs in Table 1. Note that `direct-optimization` runs very slowly, so we instead tune its regularizer weight on the first 40 test samples, select the best regularizer weight, and report its performance on the whole test set using the selected regularizer weight (more details in the appendix). The results demonstrate that our method significantly outperforms `direct-learning`, and the performance is comparable to `direct-optimization`.

Table 2: Path-planning evaluation of the average running time on the first 50 samples in the test set

	Avg. time (s)
`direct-learning`	7.96×10^{-4}
Ours	7.96×10^{-4}
`direct-optimization`	1.17×10^{4}

As a qualitative evaluation, Figure 3 shows the predictions of both `direct-learning` and our method on samples from the test set. We select the run with minimum average Chamfer distance over the test set for both `direct-learning` and our method, respectively. The results indicate that our method is better than `direct-learning` on handling details in the fiber path. We also visualize the Chamfer distance from each individual point on one path to the other, and observe that the distances for our method are generally lower than for `direct-learning`.

Running time comparison. To train needed neural networks, `direct-optimization` takes roughly 10 minutes, `direct-learning` takes roughly 1 hour, and our method takes roughly 5 days. Note that training costs are amortized, since we only need to train once. We then evaluate the inference time of the three algorithms on a server with two Intel(R) Xeon(R) E5-2699 v3 CPUs running at 2.30GHz. Since small neural networks generally run faster on the CPU, we run all of the tests solely on CPU. We run every algorithm on the first 50 paths in the test set and report the average inference time in Table 2. As we can see, both ours and `direct-learning` achieve a running time below 1 millisecond, while `direct-optimization` runs orders of magnitude slower.

Fiber path quality evaluated on a real printer. Lastly, we test our extruder path solutions on a real Markforged Mark Two 3D printer. We set the desired fiber path to a star, and print the star itself

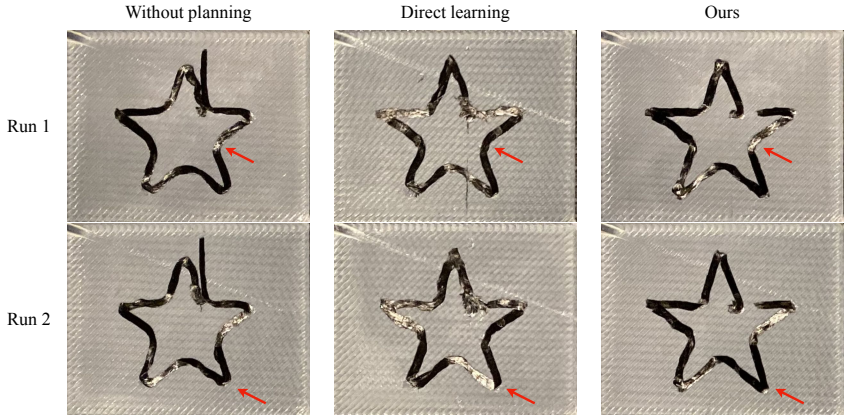


Figure 4: Path-planning evaluation of "without planning" *vs.* `direct-learning` *vs.* ours on Markforged Mark Two, with a star as the desired fiber. For "without planning", we have the same extruder path for the 2 runs; for `direct-learning` and ours, the 2 runs are predictions from neural networks trained with different initializations.

(without planning) as well as solutions from both `direct-learning` and our method. We select regularizer weights based on Table 1, *i.e.*, 0.1 for `direct-learning`, 1.5 for our method, and we show two of the trained models. The results are visualized in Figure 4, confirming that our method successfully improves the quality of the printed fiber.

6 Case study: constrained soft robot inverse kinematics

Partial differential equations (PDEs) are a powerful tool for describing complex relationships between variables and have been used widely in areas including physics, engineering, and finance. In PDE-constrained optimization [8], the goal is to optimize a cost function such that the constraints can be written as PDEs, *i.e.*, the solutions are consistent with the relationships specified by the PDE. In the context of synthesis problems, we can consider the boundary conditions of the PDE as the design, and the solution of the PDE as the realization. Similar to other synthesis problems, different boundary conditions can result in the same PDE solution, and the cost function may not have a unique minimum. In the above situations, we propose to apply our method. We test our method on a specific PDE-constrained optimization problem—constrained soft robot inverse kinematics, which serves as a representative use case of our method in this large category of problems.

Soft robots made of elastic materials, have received significant recent attention because of their reduced potential harm when working with humans [74]. Researchers have explored a variety of applications, including surgical assistance [19], bio-mimicry [61], and human-robot interaction [67]. In this case study, as in Xue et al. [97], we use a snake-like soft robot with a fixed bottom, in which we can control the stretch ratios on both sides of the robot. As shown in Figure 1d, the objective is for the midpoint at the top of the robot to reach a target, while making sure that the robot does not collide with a fixed-size circular obstacle. The relationship between the robot's shape and the stretch ratios can be written as a PDE, and as shown in Figure 1d, there are different solutions to achieve the goal.

6.1 Cost Function

We adopt the soft robot from Xue et al. [97], which has an original height of 10 and an original width of 0.5, with its bottom is fixed. The goal vector $\boldsymbol{g}$ is in $\mathbb{R}^{2\times 2}$, where $\boldsymbol{g}_1 \in \mathbb{R}^2$ indicates the target location and $\boldsymbol{g}_2 \in \mathbb{R}^2$ indicates the obstacle location. We denote the radius of the obstacle as r, which we set to 0.9. The design (control) vector $\boldsymbol{\theta}$ is in $\mathbb{R}^n$ with $n = 40$, with $\theta_i \in \mathbb{R}$ indicating the stretch ratio of the i-th segment (*e.g.*, $\theta_i = 0.95$ indicates the i-th segment is contracted by 5%). The physical realization vector $\boldsymbol{u}$ is in $\mathbb{R}^{m\times 2}$ with $m = 103$, where $\boldsymbol{u}_i \in \mathbb{R}^2$ indicates the location of the i-th vertex on the soft robot's mesh. The location of the top midpoint is denoted as $\boldsymbol{u}_{\texttt{tm}} \in \mathbb{R}^2$. Implicitly, the relationship between $\boldsymbol{\theta}$ and $\boldsymbol{u}$ obeys a PDE that governs the deformation of the robot, as detailed in Xue et al. [97].

The cost function $\mathcal{L}_{\boldsymbol{g}}(\cdot,\cdot)$ is defined as

$$\mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta},\boldsymbol{u}) \coloneqq \frac{1}{2}||\boldsymbol{g}_1 - \boldsymbol{u}_{\texttt{tm}}||_2^2 + \lambda_1 \cdot \mathcal{B}(\boldsymbol{u},\boldsymbol{g}_2) + \lambda_2 \cdot \mathcal{R}(\boldsymbol{\theta}). \tag{9}$$

The first term $||\boldsymbol{g}_1 - \boldsymbol{u}_{\texttt{tm}}||_2^2$ is the squared Euclidean distance between the top midpoint of the robot and the target. The second term, weighted by a hyper-parameter λ_1 (which we fix at 0.5), enforces

Table 3: Soft-robot evaluation of the number of successful cases (over 1,000) on test set

Regularizer weight	0.03	0.05	0.07	0.09
`direct-learning`	907.7±3.1	918.3±3.4	910.7±3.4	912.3±3.8
Ours	**986.3±0.5**	**975.0±3.9**	**981.7±5.0**	**984.7±4.5**
`direct-optimization`	997.0±0.5	998.0±0.0	998.3±0.7	997.7±0.5

Table 4: Soft-robot evaluation of the average distance to the target on successful cases on test set

Regularizer weight	0.03	0.05	0.07	0.09
`direct-learning`	0.2171±0.0016	0.2200±0.0028	0.2236±0.0006	0.2179±0.0036
Ours	**0.0657±0.0093**	**0.0464±0.0018**	**0.0599±0.0097**	**0.0691±0.0224**
`direct-optimization`	0.0233±0.0003	0.0240±0.0004	0.0241±0.0004	0.0242±0.0003

the constraint via a barrier function [65, 66] for the obstacle $\mathcal{B}(\boldsymbol{u}, \boldsymbol{g}_2)$:

$$\mathcal{B}(\boldsymbol{u}, \boldsymbol{g}_2) \coloneqq \frac{1}{m}\sum_{i=1}^{m}\big(\max(r + \Delta r - ||\boldsymbol{u}_i - \boldsymbol{g}_2||_2, 0)\big)^2, \tag{10}$$

where Δr is a hyper-parameter (which we fix at 0.1). A positive Δr provides a penalty as well as nonzero gradients when the robot gets close to the obstacle. The last term contains another hyper-parameter λ_2, varied in our experiments, weighting a smooth regularization term $\mathcal{R}(\boldsymbol{\theta})$, with

$$\mathcal{R}(\boldsymbol{\theta}) \coloneqq \frac{1}{n-4}\sum_{\substack{1<i<n, i\neq n/2,\\ i\neq n/2+1}}\left(\frac{\theta_{i+1}-\theta_i}{2} - \frac{\theta_i - \theta_{i-1}}{2}\right)^2, \tag{11}$$

where θ_i for $i = 1, 2, \cdots, n/2$ corresponds to stretch ratios on the left-hand side of the robot, and θ_i for $i = n/2+1, \cdots, n$ corresponds to stretch ratios on the right-hand side of the robot. This regularizer prevents unphysical deformations with strong discontinuities.

6.2 Evaluation metric

Since there are two objectives—"reach" and "avoid"—in this task, we have two evaluation metrics. The first metric is the number of cases that successfully avoid the obstacle (*i.e.*, have all vertex positions outside the obstacle circle). The second metric is the average Euclidean distance of the robot's top midpoint to the target for successful cases.

6.3 Implementation

Using the finite element method [44] and the code from Xue et al. [97] (MIT license), we randomly generate 40,000 data samples, and split them into 90% training, 7.5% validation, 2.5% testing. For encoder, decoder, and `direct-learning`, we use an MLP with 3 hidden layers and ReLU activation. We train every model for 200 epochs with a learning rate of 1×10^{-3} using PyTorch [68] and Adam optimizer [54]. For `direct-optimization`, we use the BFGS implementation in SciPy [92]. More implementation details are included in the appendix.

6.4 Experiments

Design quality evaluation. We experiment with different regularizer weights λ_2 for our method and the two baselines. During training, we randomly sample the location of the obstacle, and we ensure the robot never collides with the obstacle for `direct-learning`, since it does not have access to the realization vector and thus its loss function cannot contain the barrier function term for the obstacle (more details about `direct-learning` are in the appendix). For a fair comparison, during testing, we set the random seed to 0 such that for the same test sample, the obstacle will appear at the same location for all algorithms. The number of cases in which the robot successfully avoids the obstacle, with standard error for 3 runs, is shown in Table 3. The average Euclidean distance to the target for successful cases, with its standard error, is shown in Table 4. As the numbers show, our method is competitive to `direct-optimization`, and performs much better than `direct-learning`. Samples from the test set are shown in Figure 5 (for both algorithms, we select the best run with a regularizer weight of 0.5). We can see that our method collides less frequently while reaching the target more accurately than `direct-learning`.

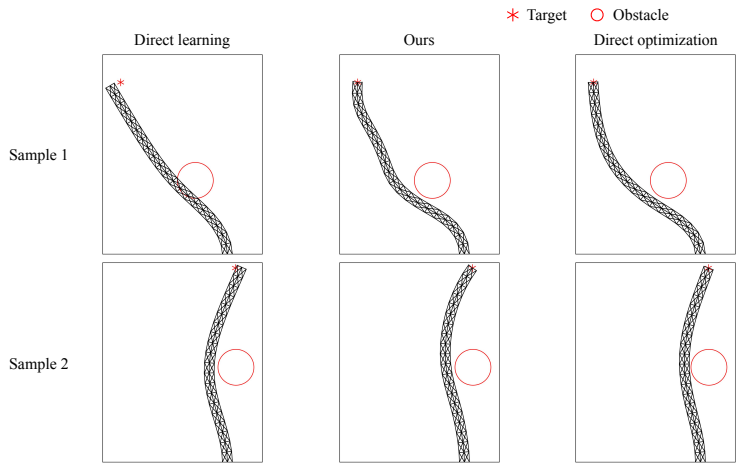


Figure 5: Soft-robot evaluation of `direct-learning` *vs.* ours *vs.* `direct-optimization` on examples from the test set. The direct learning baseline both violates the constraints (Sample 1) and fails to reach the target (Samples 1 and 2), while the run time of `direct-optimization` is 4000 times that of our method.

Running time. To train the neural networks, `direct-optimization` takes roughly 2 hours, `direct-learning` takes roughly 4 hours, and our method takes roughly 4.5 hours. Note that training costs are amortized, since we only need to train once. We then test all algorithms on a server with two Intel(R) Xeon(R) E5-2699 v3 CPUs running at 2.30GHz. Everything runs solely on the CPU, maximizing efficiency for the small neural networks we use. The results are shown in Table 5, showing that we successfully reduce the running time from over 1 second to less than 1 millisecond. Note that the soft robot is relatively small (40 control variables), and the time complexity of BFGS grows quadratically w.r.t. the number of parameters. Therefore, for more complex soft robots or PDE-constrained optimization problems with a larger number of variables, the running time advantage of our method can be of even greater importance.

Table 5: Soft-robot evaluation of the average running time on the test set

	Avg. time (s)
`direct-learning`	3.12×10^{-4}
Ours	3.34×10^{-4}
`direct-optimization`	1.33×10^{0}

7 Discussion

In this work, we provided an amortized approach to synthesis problems in machine learning. To tackle the non-differentiability of physical system realizations, the huge computational cost of realization processes, and the non-uniqueness of the design solution, we designed a two-stage neural network architecture, where we first learn the decoder, a surrogate that approximates the realization processes, and then learn the encoder, which proposes a design for an input goal. We tested our approach on two case studies on fiber extruder path planning and constrained soft robot inverse kinematics, where we demonstrated that our method provides designs with much higher quality than supervised learning of the design problem, while being competitive in quality to and orders of magnitude faster than direct optimization of the design solution.

Although the experiments in both case studies show the effectiveness of our approach, we would like to mention some limitations of our method. First, to effectively learn a differentiable surrogate for the realization process, we need to be able to generate a substantial number of viable designs. We also need a simulator to calculate physical realizations of them, and the realization process has to be deterministic, although extensions might consider probability distributions over realizations. Also, to train the encoder, we need the objective ("goal") to be quantifiable. Our method provides the greatest gains if the realization is computationally expensive and/or non-differentiable, or if our encoder can exploit the non-uniqueness of designs to choose one good option where supervised learning would have learned a poor "average" solution. Additionally, due to the cost of neural network training, amortization is only a good idea when we need to solve one design problem many times with different goals, or we need fast inference. From a societal point of view, the primary negative consequence is the potential for replacing human labor in design. We view the present approach, however, as part of larger human-in-the-loop design processes in line with other software tools for modeling and fabrication.

Acknowledgements

We would like to thank Geoffrey Roeder for helping set up the 3D printer, Amit Bermano, Jimmy Wu, and members of the Princeton Laboratory for Intelligent Probabilistic Systems for valuable discussions and feedback, as well as Markforged. This work is partially supported by the Princeton School of Engineering and Applied Science, as well as the U. S. National Science Foundation under grants #IIS-1815070 and #IIS-2007278.

References

[1] Suleman Asif. *Modelling and path planning for additive manufacturing of continuous fiber composites*. PhD thesis, 2018.

[2] Han Bao, Clay Scott, and Masashi Sugiyama. Calibrated surrogate losses for adversarially robust classification. In *Conference on Learning Theory*, pages 408–451. PMLR, 2020.

[3] Harry G Barrow, Jay M Tenenbaum, Robert C Bolles, and Helen C Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *IJCAI*, 1977.

[4] Cenk Baykal and Ron Alterovitz. Asymptotically optimal design of piecewise cylindrical robots using motion planning. In *Robotics: Science and Systems*, volume 2017, 2017.

[5] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

[6] Atharv Bhosekar and Marianthi Ierapetritou. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering*, 108:250–267, 2018.

[7] V Bhuvaneswari, M Priyadharshini, C Deepa, D Balaji, L Rajeshkumar, and M Ramesh. Deep learning for material synthesis and manufacturing systems: a review. *Materials Today: Proceedings*, 2021.

[8] Lorenz T Biegler, Omar Ghattas, Matthias Heinkenschloss, and Bart van Bloemen Waanders. Large-scale pde-constrained optimization: an introduction. In *Large-Scale PDE-Constrained Optimization*, pages 3–13. Springer, 2003.

[9] David Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. In *International conference on machine learning*, pages 773–782. PMLR, 2019.

[10] David H Brookes and Jennifer Listgarten. Design by adaptive sampling. *arXiv preprint arXiv:1810.03714*, 2018.

[11] JA Cabrera, A Simon, and M Prado. Optimal synthesis of mechanisms with genetic algorithms. *Mechanism and machine theory*, 37(10):1165–1177, 2002.

[12] JA Cabrera, A Ortiz, F Nadal, and JJ Castillo. An evolutionary algorithm for path synthesis of mechanisms. *Mechanism and Machine Theory*, 46(2):127–141, 2011.

[13] Thais Campos and Hadas Kress-Gazit. Synthesizing modular manipulators for tasks with time, obstacle, and torque constraints. *arXiv preprint arXiv:2106.09487*, 2021.

[14] Thais Campos, Jeevana Priya Inala, Armando Solar-Lezama, and Hadas Kress-Gazit. Task-based design of ad-hoc modular manipulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6058–6064. IEEE, 2019.

[15] Thais Campos de Almeida, Samhita Marri, and Hadas Kress-Gazit. Automated synthesis of modular manipulators' structure and control for continuous tasks around obstacles. *Robotics: Science and Systems 2020*, 2020.

[16] Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 23(6):1241–1250, 2018.

[17] I-Ming Chen and Joel W Burdick. Determining task optimal modular robot assembly configurations. In *proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 1, pages 132–137. IEEE, 1995.

[18] Wan Kyun Chung, Jeongheon Han, Youngil Youm, and SH Kim. Task based design of modular robot manipulator using efficient genetic algorithm. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 507–512. IEEE, 1997.

[19] Matteo Cianchetti, Tommaso Ranzani, Giada Gerboni, Thrishantha Nanayakkara, Kaspar Althoefer, Prokar Dasgupta, and Arianna Menciassi. Soft robotics technologies to address shortcomings in today's minimally invasive surgery: the stiff-flop approach. *Soft robotics*, 1 (2):122–131, 2014.

[20] Connor W Coley, William H Green, and Klavs F Jensen. Machine learning in computer-aided synthesis planning. *Accounts of chemical research*, 51(5):1281–1289, 2018.

[21] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013.

[22] Erwin Coumans. Bullet physics engine. *Open Source Software: http://bulletphysics. org*, 2010.

[23] Anubhav Dogra, Srikant Sekhar Padhee, and Ekta Singla. An optimal architectural design for unconventional modular reconfigurable manipulation system. *Journal of Mechanical Design*, 143(6):063303, 2021.

[24] Jesse Engel, Matthew Hoffman, and Adam Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. In *International Conference on Learning Representations*, 2018.

[25] Michael Everett, Yu Fan Chen, and Jonathan P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059. IEEE, 2018.

[26] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, 2017.

[27] Clara Fannjiang and Jennifer Listgarten. Autofocused oracles for model-based design. *Advances in Neural Information Processing Systems*, 33, 2020.

[28] Shane Farritor, Steven Dubowsky, Nathan Rutman, and Jeffrey Cole. A systems-level modular design approach to field robotics. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 4, pages 2890–2895. IEEE, 1996.

[29] Alexander IJ Forrester and Andy J Keane. Recent advances in surrogate-based optimization. *Progress in aerospace sciences*, 45(1-3):50–79, 2009.

[30] Justin Fu and Sergey Levine. Offline model-based optimization via normalized maximum likelihood estimation. In *International Conference on Learning Representations*, 2020.

[31] Nuwan Ganganath, Chi-Tsun Cheng, Kai-Yin Fok, and K Tse Chi. Trajectory planning for 3d printing: A revisit to traveling salesman problem. In *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, pages 287–290. IEEE, 2016.

[32] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel 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, 2018.

[33] Josif Grabocka, Randolf Scholz, and Lars Schmidt-Thieme. Learning surrogate losses. *arXiv preprint arXiv:1905.10108*, 2019.

[34] Yanran Guan, Han Liu, Kun Liu, Kangxue Yin, Ruizhen Hu, Oliver van Kaick, Yan Zhang, Ersin Yumer, Nathan Carr, Radomir Mech, and Hao Zhang. FAME: 3d shape generation via functionality-aware model evolution. *IEEE Transactions on Visualization and Computer Graphics*, 2020.

[35] Sumit Gulwani. Program synthesis. *Software Systems Safety*, pages 43–75, 2014.

[36] Anvita Gupta and James Zou. Feedback gan for dna optimizes protein functions. *Nature Machine Intelligence*, 1(2):105–111, 2019.

[37] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Task-based limb optimization for legged robots. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2062–2068. IEEE, 2016.

[38] Sehoon Ha, Stelian Coros, Alexander Alspach, James M Bern, Joohyung Kim, and Katsu Yamane. Computational design of robotic devices from high-level motion specifications. *IEEE Transactions on Robotics*, 34(5):1240–1251, 2018.

[39] Zhong-Hua Han, Ke-Shi Zhang, et al. Surrogate-based optimization. *Real-world applications of genetic algorithms*, 343, 2012.

[40] Steve Hanneke, Liu Yang, et al. Surrogate losses in passive and active learning. *Electronic Journal of Statistics*, 13(2):4646–4708, 2019.

[41] Alex Hawkins-Hooker, Florence Depardieu, Sebastien Baur, Guillaume Couairon, Arthur Chen, and David Bikard. Generating functional protein variants with variational autoencoders. *PLoS computational biology*, 17(2):e1008736, 2021.

[42] Gregory S Hornby, Hod Lipson, and Jordan B Pollack. Evolution of generative design systems for modular physical robots. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 4, pages 4146–4151. IEEE, 2001.

[43] Jiaqi Huang, Qian Chen, Hao Jiang, Bin Zou, Lei Li, Jikai Liu, and Huangchao Yu. A survey of design methods for material extrusion polymer 3d printing. *Virtual and Physical Prototyping*, 15(2):148–162, 2020.

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

[45] Haoyan Huo, Ziqin Rong, Olga Kononova, Wenhao Sun, Tiago Botari, Tanjin He, Vahe Tshitoyan, and Gerbrand Ceder. Semi-supervised machine-learning classification of materials synthesis procedures. *npj Computational Materials*, 5(1):1–7, 2019.

[46] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Multi-objective molecule generation using interpretable substructures. In *International Conference on Machine Learning*, pages 4849–4859. PMLR, 2020.

[47] Wengong Jin, Jonathan M Stokes, Richard T Eastman, Zina Itkin, Alexey V Zakharov, James J Collins, Tommi S Jaakkola, and Regina Barzilay. Deep learning identifies synergistic drug combinations for treating covid-19. *Proceedings of the National Academy of Sciences*, 118 (39), 2021.

[48] Nikos Ath Kallioras, Georgios Kazakis, and Nikos D Lagaros. Accelerated topology optimization by means of deep learning. *Structural and Multidisciplinary Optimization*, 62(3): 1185–1212, 2020.

[49] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[50] M Khorshidi, M Soheilypour, M Peyro, A Atai, and M Shariat Panahi. Optimal design of four-bar mechanisms using a hybrid multi-objective ga with adaptive local search. *Mechanism and Machine Theory*, 46(10):1453–1465, 2011.

[51] Nathan Killoran, Leo J Lee, Andrew Delong, David Duvenaud, and Brendan J Frey. Generating and designing dna with deep generative models. *arXiv preprint arXiv:1712.06148*, 2017.

[52] Edward Kim, Kevin Huang, Adam Saunders, Andrew McCallum, Gerbrand Ceder, and Elsa Olivetti. Materials synthesis insights from scientific literature via text extraction and machine learning. *Chemistry of Materials*, 29(21):9436–9444, 2017.

[53] J-O Kim and Pradeep K Khosla. A formulation for task based design of robot manipulators. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, volume 3, pages 2310–2317. IEEE, 1993.

[54] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[55] Hunter T Kollmann, Diab W Abueidda, Seid Koric, Erman Guleryuz, and Nahil A Sobh. Deep learning for topology optimization of 2d metamaterials. *Materials & Design*, 196:109098, 2020.

[56] Slawomir Koziel and Leifur Leifsson. *Surrogate-based modeling and optimization*. Springer, 2013.

[57] Slawomir Koziel and Stanislav Ogurtsov. Surrogate-based optimization. In *Antenna Design by Simulation-Driven Optimization*, pages 13–24. Springer, 2014.

[58] Slawomir Koziel, David Echeverría Ciaurri, and Leifur Leifsson. Surrogate-based methods. In *Computational optimization, methods and algorithms*, pages 33–59. Springer, 2011.

[59] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[60] Chris Leger and John Bares. Automated task-based synthesis and optimization of field robots. 1999.

[61] Guorui Li, Xiangping Chen, Fanghao Zhou, Yiming Liang, Youhua Xiao, Xunuo Cao, Zhen Zhang, Mingqi Zhang, Baosheng Wu, Shunyu Yin, et al. Self-powered soft robot in the mariana trench. *Nature*, 591(7848):66–71, 2021.

[62] Lanlan Liu, Mingzhe Wang, and Jia Deng. A unified framework of surrogate loss by refactoring and interpolation. In *ECCV*, 2020.

[63] Iain Murray, Ryan Adams, and David MacKay. Elliptical slice sampling. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 541–548. JMLR Workshop and Conference Proceedings, 2010.

[64] Gattigorla Nagendar, Digvijay Singh, Vineeth N Balasubramanian, and CV Jawahar. Neuro-iou: Learning a surrogate loss for semantic segmentation. In *BMVC*, page 278, 2018.

[65] Yurii Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

[66] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[67] Gaoyang Pang, Geng Yang, Wenzheng Heng, Zhiqiu Ye, Xiaoyan Huang, Hua-Yong Yang, and Zhibo Pang. Coboskin: Soft robot skin with variable stiffness for safer human–robot collaboration. *IEEE Transactions on Industrial Electronics*, 68(4):3303–3314, 2020.

[68] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[69] Sarosh Patel and Tarek Sobh. Task based synthesis of serial manipulators. *Journal of advanced research*, 6(3):479–492, 2015.

[70] Yash Patel, Tomáš Hodaň, and Jiří Matas. Learning surrogates via deep embedding. In *European Conference on Computer Vision*, pages 205–221. Springer, 2020.

[71] Daniil Polykovskiy, Alexander Zhebrak, Dmitry Vetrov, Yan Ivanenkov, Vladimir Aladinskiy, Polina Mamoshina, Marine Bozdaganyan, Alexander Aliper, Alex Zhavoronkov, and Artur Kadurin. Entangled conditional adversarial autoencoder for de novo drug discovery. *Molecular pharmaceutics*, 15(10):4398–4405, 2018.

[72] Alex Renda, Yishen Chen, Charith Mendis, and Michael Carbin. Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 442–455. IEEE, 2020.

[73] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), 2021.

[74] Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521 (7553):467–475, 2015.

[75] Hidenori Sasaki and Hajime Igarashi. Topology optimization accelerated by deep learning. *IEEE Transactions on Magnetics*, 55(6):1–5, 2019.

[76] Ari Seff, Wenda Zhou, Farhan Damani, Abigail Doyle, and Ryan P Adams. Discrete object generation with reversible inductive construction. In *Advances in Neural Information Processing Systems*, volume 32, pages 10353–10363, 2019.

[77] Nurhalida Shahrubudin, Te Chuan Lee, and Rhaizan Ramlan. An overview on 3d printing technology: technological, materials, and applications. *Procedia Manufacturing*, 35:1286–1296, 2019.

[78] Aniruddha V Shembekar, Yeo Jung Yoon, Alec Kanyuck, and Satyandra K Gupta. Trajectory planning for conformal 3d printing using non-planar layers. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 51722, page V01AT02A026. American Society of Mechanical Engineers, 2018.

[79] Sergey Shirobokov, Vladislav Belavin, Michael Kagan, Andrei Ustyuzhanin, and Atilim Gunes Baydin. Black-box optimization with local generative surrogates. In *Workshop on Real World Experiment Design and Active Learning at International Conference on Machine Learning*, 2020.

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

[81] Satinder P Singh, Andrew G Barto, Roderic Grupen, Christopher Connolly, et al. Robust reinforcement learning in motion planning. *Advances in neural information processing systems*, pages 655–655, 1994.

[82] Suwin Sleesongsom and Sujin Bureerat. Four-bar linkage path generation through self-adaptive population size teaching-learning based optimization. *Knowledge-Based Systems*, 135:180–191, 2017.

[83] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

[84] Enrico Stragiotti. *Continuous Fiber Path Planning Algorithm for 3D Printed Optimal Mechanical Properties.* PhD thesis, Politecnico di Torino, 2020.

[85] Devika Subramanian et al. Kinematic synthesis with configuration spaces. *Research in Engineering Design*, 7(3):193–213, 1995.

[86] Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B Tenenbaum, and William T Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[87] Saleh Tabandeh, William Melek, Mohammad Biglarbegian, Seong-hoon Peter Won, and Chris Clark. A memetic algorithm approach for solving the task-based configuration optimization problem in serial modular and reconfigurable robots. *Robotica*, 34(9):1979–2008, 2016.

[88] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations*, 2018.

[89] 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.

[90] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, et al. Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery*, 18(6): 463–477, 2019.

[91] EJ Van Henten, DA Van't Slot, CWJ Hol, and LG Van Willigenburg. Optimal manipulator design for a cucumber harvesting robot. *Computers and electronics in agriculture*, 65(2): 247–257, 2009.

[92] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[93] Michael Yu Wang, Xiaoming Wang, and Dongming Guo. A level set method for structural topology optimization. *Computer methods in applied mechanics and engineering*, 192(1-2): 227–246, 2003.

[94] Julian Whitman and Howie Choset. Task-specific manipulator design and trajectory synthesis. *IEEE Robotics and Automation Letters*, 4(2):301–308, 2018.

[95] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T Freeman, and Joshua B Tenenbaum. MarrNet: 3D Shape Reconstruction via 2.5D Sketches. In *NIPS*, 2017.

[96] Hong Xiao, Wei Han, Wenbin Tang, and Yugang Duan. An efficient and adaptable path planning algorithm for automated fiber placement based on meshing and multi guidelines. *Materials*, 13(18):4209, 2020.

[97] Tianju Xue, Alex Beatson, Sigrid Adriaenssens, and Ryan Adams. Amortized finite element analysis for fast pde-constrained optimization. In *International Conference on Machine Learning*, pages 10638–10647. PMLR, 2020.

[98] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B Tenenbaum. Neural-symbolic vqa: disentangling reasoning from vision and language understanding. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 1039–1050, 2018.

[99] Yonggyun Yu, Taeil Hur, and Jaeho Jung. Deep learning for topology optimization design. *arXiv preprint arXiv:1801.05463*, 2018.

[100] Zhuoning Yuan, Yan Yan, Milan Sonka, and Tianbao Yang. Robust deep auc maximization: A new surrogate loss and empirical studies on medical image classification. *arXiv preprint arXiv:2012.03173*, 2020.

[101] Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided mechanical toy modeling. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.

A Details about extruder path planning

A.1 Methods

Ours. As we discussed before in Section 3, we train a decoder (surrogate) using Equation 2 and an encoder using Equation 3, with the cost function defined in Equation 6. We take the trained encoder $\phi^*(\cdot)$ as our final model in use.

`direct-learning`. As we discussed in Section 4, we train `direct-learning` using Equation 4, with a regularizer as defined in Equation 7. Note that this is equivalent to training to minimize the cost function $\mathcal{L}_{\cdot}(\cdot,\cdot)$ in Equation 6.

`direct-optimization`. As described in Section 4, we build `direct-optimization` as in Equation 5, with a trained surrogate of the physical realization process. Here, to enforce that points $\boldsymbol{\theta}_i$ are evenly distributed along the extruder path, we use a slightly different cost function $\mathcal{L}_{\mathsf{do},\cdot}(\cdot,\cdot)$:

$$\mathcal{L}_{\mathsf{do},\boldsymbol{g}}(\boldsymbol{\theta},\boldsymbol{u}) \coloneqq d_{\mathsf{do}}(\boldsymbol{g},\boldsymbol{u}) + \lambda_{\mathsf{do}} \cdot \mathcal{R}_{\mathsf{do}}(\boldsymbol{\theta}), \tag{12}$$

where we have a distance function $d_{\mathsf{do}}(\cdot,\cdot)$ and a smooth regularizer $\mathcal{R}_{\mathsf{do}}(\cdot)$. The smooth regularizer $\mathcal{R}_{\mathsf{do}}(\cdot)$ is derived from Equation 7 by requiring $||\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i||_2$ to be the same for all i:

$$\mathcal{R}_{\mathsf{do}}(\boldsymbol{\theta}) \coloneqq \frac{1}{S_{\boldsymbol{\theta}}} \sum_{i=2}^{n-1} \left(\frac{\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i}{2} - \frac{\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}}{2} \right)^2, \tag{13}$$

where $S_{\boldsymbol{\theta}}$ is the length of the extruder path $\boldsymbol{\theta}$; this intrinsically enforces points in $\boldsymbol{\theta}$ to be evenly spaced. To measure the distance between $\boldsymbol{g}$ and $\boldsymbol{u}$, we first map them into two functions $\boldsymbol{f}_{\boldsymbol{g}}(\cdot)$ and $\boldsymbol{f}_{\boldsymbol{u}}(\cdot)$, such that $\boldsymbol{f}_{\boldsymbol{g}}(s) \in \mathbb{R}^2$ is the location if we walk a distance s along the path $\boldsymbol{g}$ (assuming the path is piecewise linear), and similarly for $\boldsymbol{f}_{\boldsymbol{u}}(\cdot)$. Then the distance function is defined as

$$d_{\mathsf{do}}(\boldsymbol{g},\boldsymbol{u}) \coloneqq \int_0^1 \left\| \boldsymbol{f}_{\boldsymbol{g}}(x \cdot S_{\boldsymbol{g}}) - \boldsymbol{f}_{\boldsymbol{u}}(x \cdot S_{\boldsymbol{u}}) \right\|^2 \mathrm{d}x, \tag{14}$$

where $S_{\boldsymbol{g}}$ and $S_{\boldsymbol{u}}$ denote the lengths of paths $\boldsymbol{g}$ and $\boldsymbol{u}$, respectively.

A.2 Data generation

Random curve generation. To build the dataset, we first need to generate some random 2D curves, which can be used as extruder paths later. The curves should be smooth and non-intersecting. For each path, we take both the x and y to be Gaussian processes whose kernel function $K(\cdot,\cdot)$ is

$$K(x,x') \coloneqq \exp\left(-\frac{\sin^2\big((x-x')/2\big)}{2\,l^2} \right), \tag{15}$$

with the two axes independent and $l = 0.1$. We use elliptical slice sampling [63] (New BSD License): for each path, we start from 1,000 points on a unit circle, and sample 1,000 times. To avoid intersections, we use a log-likelihood of $-\infty$ for a self-intersecting path, and a log-likelihood of 0 for a non-intersecting path. We generate 10,000 paths using this approach.

Simulator. Since it is time-consuming to print every extruder path we generate in the last step on a real printer, we build a simulation system by using Bullet [22] to help us generate fiber paths. The simulator is also used in our evaluation. We calibrate the simulator on two materials—carbon fiber and Kevlar, respectively. To calibrate, we print paths shaped as sine functions with amplitudes ranging from 3.0 cm to 6.5 cm on the Markforged Mark Two printer (Figure 6). We then measure the amplitudes of the printed fiber paths, which will be lower than the amplitudes of the extruder paths because of smoothing, and fit lines to actual amplitude vs. extruder amplitude (Figure 7a). After that, we perform a grid search on the parameters of the simulator (such as stiffness and friction), run simulations with the sine functions as extruder paths, and collect data pairs of extruder path amplitudes and simulated fiber path amplitudes. We end up with calibrated simulators for carbon fiber (Figure 7b) and Kevlar (Figure 7c), by selecting the set of parameters for each having the minimum sum of squared distances between the fitted line from Figure 7a) and the collected simulation data points. Finally, we run the tuned simulators on the generated extruder paths. Though we have conducted experiments with both materials, due to space constraints, the experiments in the paper use data generated from the carbon fiber simulator. We visualize some extruder paths and simulated carbon fiber paths in Figure 8.

A.3 Mapping from a path to another

We have to design a neural network that can take in an input path ($\mathbb{R}^{n\times 2}$) and output another path ($\mathbb{R}^{n\times 2}$), which can be used for encoder, decoder, and `direct-learning`. Both paths are sequences

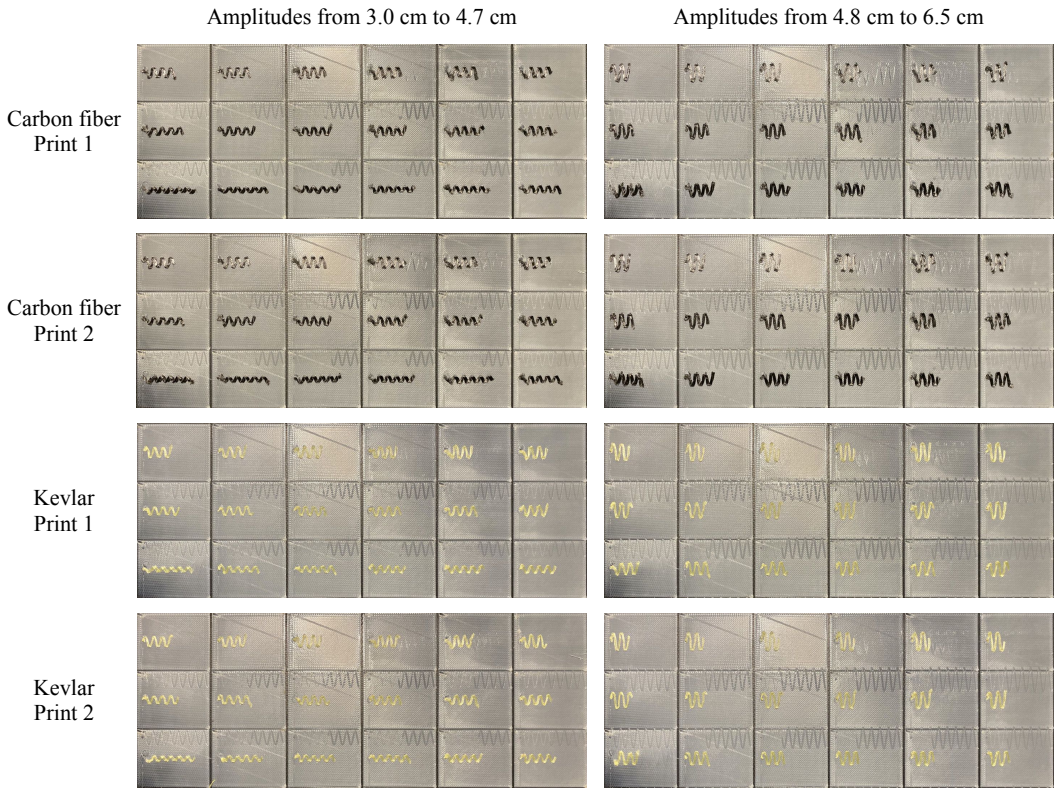
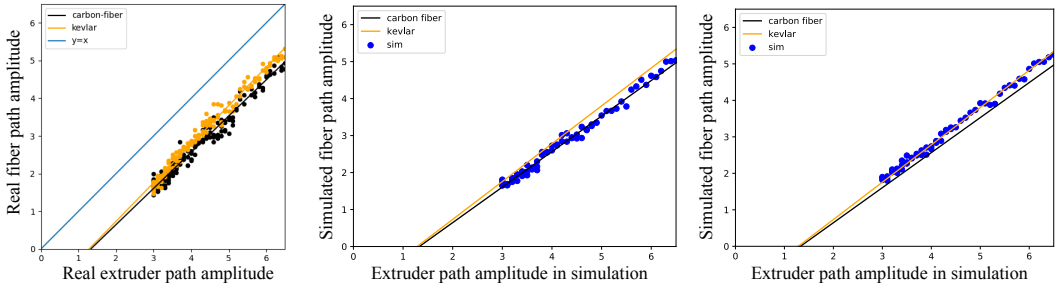


Figure 6: We print paths shaped as sine functions with amplitudes from 3.0 cm to 6.5 cm on the Markforged Mark Two printer, using carbon fiber and Kevlar, respectively. We print everything twice.



a) lines fitted on real data b) simulator tuned for carbon fiber c) simulator tuned for Kevlar

Figure 7: (a) We print paths shaped as sine functions with different amplitudes, collect amplitudes of the printed fiber paths, and fit lines through the data we collected. We experiment with two materials—carbon fiber and Kevlar, and the identity line is also visualized. (b) and (c) We select two sets of simulator hyper-parameters with their simulation results closest to the lines we get from the previous step for carbon fiber and Kevlar, respectively.

of n 2D coordinates. For the purpose of illustration, we use decoder as an example here. Now the input is the extruder path $\boldsymbol{\theta}$, and the output is the resulting fiber path $\boldsymbol{u}$. Remember $\boldsymbol{\theta}_i \in \mathbb{R}^2$ is the i-th row of $\boldsymbol{\theta}$. Due to the intrinsic equivariant property of the problem, one natural idea is to have a neural network that takes in a certain number of points near $\boldsymbol{\theta}_i$ in $\boldsymbol{\theta}$ and outputs the corresponding point in $\boldsymbol{u}$ (*i.e.*, $\boldsymbol{u}_i$), and we iterate over every i, as a window sliding over $\boldsymbol{\theta}$. Note that we used the same neural network for all i's.

We thus use a multilayer perceptron (MLP), which takes $2m + 1$ points (we set $m = 30$) and outputs one point. We take $\boldsymbol{\theta}_i$ as the starting point and resample m points both forward and backward along the path $\boldsymbol{\theta}$. To be specific, as in § A.1, we first map $\boldsymbol{\theta}$ into a function $\boldsymbol{f}_{\boldsymbol{\theta}}(\cdot)$ such that $\boldsymbol{f}_{\boldsymbol{\theta}}(s) \in \mathbb{R}^2$ is the location if we start from $\boldsymbol{\theta}_1$ and walk a length of s on the path. We further set $\boldsymbol{f}_{\boldsymbol{\theta}}(s) \coloneqq \boldsymbol{f}_{\boldsymbol{\theta}}(0)$ for $s < 0$ and $\boldsymbol{f}_{\boldsymbol{\theta}}(s) \coloneqq \boldsymbol{f}_{\boldsymbol{\theta}}(S_{\boldsymbol{\theta}})$ for $s > S_{\boldsymbol{\theta}}$, where $S_{\boldsymbol{\theta}}$ is the length of extruder path $\boldsymbol{\theta}$. We denote the distance of walking from $\boldsymbol{\theta}_1$ to $\boldsymbol{\theta}_i$ as s_i, *i.e.*, $\boldsymbol{f}_{\boldsymbol{\theta}}(s_i) = \boldsymbol{\theta}_i$. The input to the MLP is:

$$[\boldsymbol{f}_{-m}, \boldsymbol{f}_{-m+1}, \cdots, \boldsymbol{f}_0, \cdots, \boldsymbol{f}_m]^\mathsf{T}, \tag{16}$$

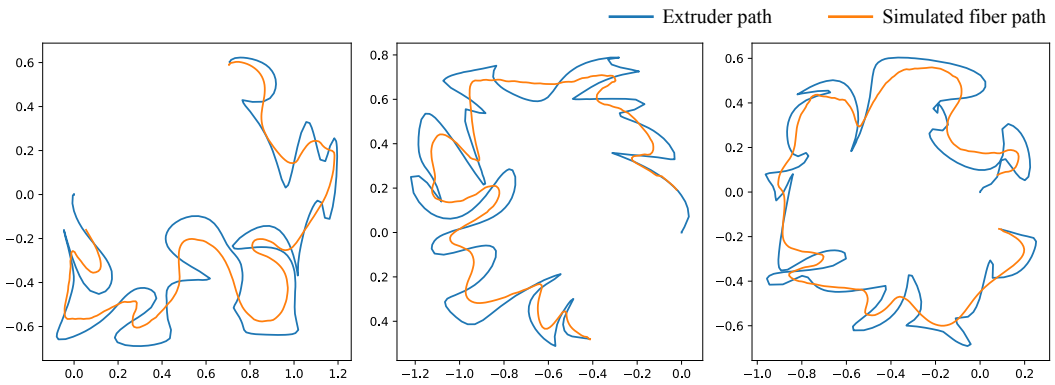


Figure 8: Samples from our dataset. We plot both the extruder paths and the simulated carbon fiber paths.

Table 6: The architecture of the MLP used in extruder path planning

Type	Configurations
Fully connected	$4m$+2 to 500
ReLU	N/A
Fully connected	500 to 200
ReLU	N/A
Fully connected	200 to 100
ReLU	N/A
Fully connected	100 to 50
ReLU	N/A
Fully connected	50 to 25
ReLU	N/A
Fully connected	25 to 2

where

$$\boldsymbol{f}_i \coloneqq \boldsymbol{f}_{\boldsymbol{\theta}}(s_i + i \cdot s_0) - \boldsymbol{f}_{\boldsymbol{\theta}}(s_i), \tag{17}$$

and $s_0 = 0.03$ is the step size. Since the problem is intrinsically translation-equivariant, we normalize every $\boldsymbol{f}_i$ by subtracting $\boldsymbol{f}_{\boldsymbol{\theta}}(s_i)$, as shown in the above equation.

A.4 Hyper-parameters and neural network training

The architecture of the MLP is shown in Table 6, and we implement it in PyTorch [68]. We coarsely tuned the architecture, including the number of hidden layers (from 1 to 6) and the size of each hidden layer. We noticed that the accuracy is not largely affected by the architecture, as long as there is at least one hidden layer. We split the dataset into 90% training (9,000 paths), 5% validation (500 paths), and 5% testing (500 paths), and we use the Adam optimizer [54] with a learning rate of 1×10^{-3}, a learning rate exponential decay of 0.95 per epoch, and a batch size of 1 (path). We train every model—the decoder, the encoder, and `direct-learning`—for 10 epochs. We use our internal cluster with 7 servers with 14 Intel(R) Xeon(R) CPUs. For our method and `direct-learning`, we train them with different regularizer weights λ = 0.1, 0.3, 0.6, 1.0, and 1.5. For `direct-optimization`, we use the BFGS implementation in SciPy [92], with a gradient tolerance of 1×10^{-7}. Since its running time is extremely long, we tune its regularizer weight on the first 40 test samples (Table 7) and select the one with the best performance. To train the needed neural networks, it takes approximately 10 minutes for `direct-optimization`, approximately 1 hour for `direct-learning`, and approximately 5 days for our method. Note that we only need to train once so that the training costs are amortized.

B Details about constrained soft robot inverse kinematics

B.1 Robot setting

We adopt the snake-like soft robot that was used in Xue et al. [97]. The robot has an original height of 10 and an original width of 0.5, and its bottom is fixed. We can control the stretch ratio of 40 segments (20 on the left-hand side and 20 on the right-hand side), as visualized in colors in Figure 9.

Table 7: Path-planning evaluation of `direct-optimization` of the average Chamfer distance on the first 40 samples in the test set evaluated in simulation

Regularizer weight	0.0001	0.0003	0.0006	0.001
`direct-optimization`	0.0171	0.0161	**0.0153**	0.0167

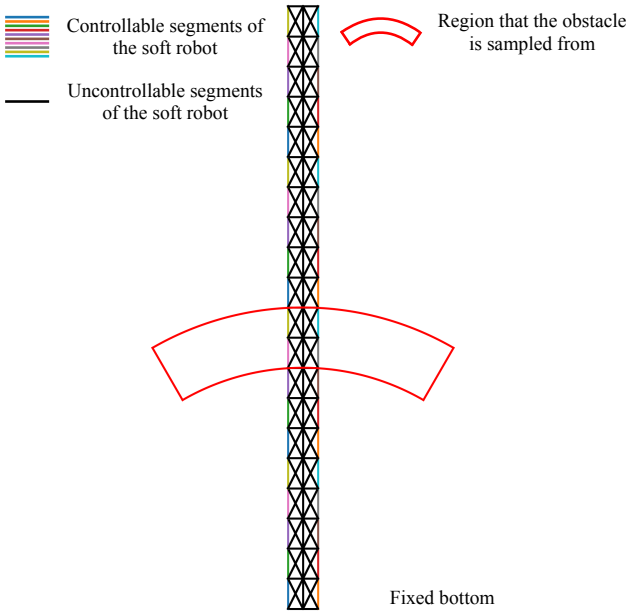


Figure 9: We visualize the soft robot with controllable segments in color, uncontrollable segments in black. The red region shows where we sample the center of the obstacle—a sector region with an angle of 60°, inner and outer radiuses of 4 and 5, respectively.

The stretch ratios are restricted to be between 0.8 and 1.2. The physical realization of the robot consists of the locations of its 103 vertexes, as shown in Figure 9.

B.2 Methods

Ours. We follow Section 3 with the cost function defined as in Equation 9, and the obstacle location is randomly sampled.

`direct-learning`. We follow Section 4 with $\mathcal{R}_{\texttt{dl}}(\cdot)$ defined as in Equation 11. Note that since we do not have access to the physical realization $\boldsymbol{u}$, we cannot have a barrier function term for the obstacle. Thus, for `direct-learning`, we still randomly sample the obstacle location, but we guarantee during training, the obstacle does not collide with the robot in every specific training sample.

`direct-optimization`. We follow Section 4 with the cost function defined as in Equation 9 and the obstacle location randomly sampled. Note that this baseline is similar to the approach used in Xue et al. [97]. The major difference is that we train the surrogate using supervised loss (as shown in Equation 2), and Xue et al. [97] trained their surrogate using a physically informed loss that minimizes the total potential energy.

B.3 Data generation

We first randomly sample the design vector $\boldsymbol{\theta}$ with each dimension i.i.d. uniformly between 0.8 and 1.2. For each design vector $\boldsymbol{\theta}$, we solve the governing PDE with the finite element method [44] to obtain the corresponding physical realization $\boldsymbol{u}$ of the robot. Note that the obstacle location is randomly sampled during training and randomly sampled with a fixed random seed (we set to 0) during testing. The center of the obstacle is uniformly sampled from a sector region, as shown in Figure 9, with an angle of 60°, an inner radius of 4, and an outer radius of 5. We altogether generate 40,000 data samples.

Table 8: The architecture of the MLP used in constrained soft robot inverse kinematics

`direct-learning`		Decoder		Encoder	
Type	Configurations	Type	Configurations	Type	Configurations
Fully connected	4 to 128	Fully connected	40 to 128	Fully connected	4 to 128
ReLU	N/A	ReLU	N/A	ReLU	N/A
Fully connected	128 to 256	Fully connected	128 to 256	Fully connected	128 to 256
ReLU	N/A	ReLU	N/A	ReLU	N/A
Fully connected	256 to 128	Fully connected	256 to 128	Fully connected	256 to 128
ReLU	N/A	ReLU	N/A	ReLU	N/A
Fully connected	128 to 40	Fully connected	128 to 206	Fully connected	128 to 40
Sigmoid	N/A	/	/	Sigmoid	N/A
Linear map	$0.2(2x-1)$	/	/	Linear map	$0.2(2x-1)$

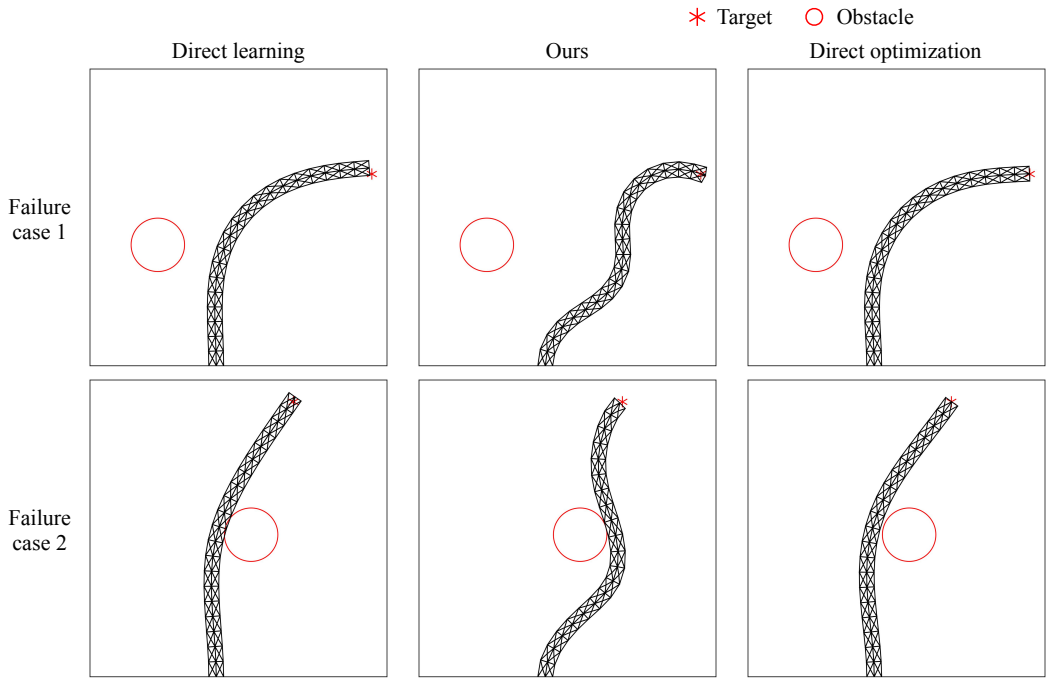


Figure 10: Failure cases of our method in test set on soft robot. In rare cases, our method might miss the target by a short distance (Samples 1 and 2) or touch the obstacle (Sample 2).

B.4 Hyper-parameters and neural network training

We use MLP for all models (encoder, decoder, and `direct-learning`) with ReLU as the activation function and 3 hidden layers of sizes 128, 256, 128, respectively (Table 8). We coarsely tuned the architecture, including the number of hidden layers (from 1 to 4) and the size of each hidden layer. Similarly, we noticed that the accuracy is not largely affected by the architecture, as long as there is at least one hidden layer. Note that for the input and output of the neural network, we subtract 1 from all stretch ratios such that they are always between -0.2 and 0.2, and we use displacement of each vertex rather than its absolute location since displacement values are mostly centered around 0. In addition, to ensure that the encoder and `direct-learning` always output stretch ratios (minus one) between -0.2 and 0.2, we apply a sigmoid layer at the end of both the encoder and `direct-learning`, and linearly map the sigmoid output to be between -0.2 and 0.2 (as in Table 8). We use the same trick in `direct-optimization` to ensure the stretch ratios never fall out of range.

We implement all neural networks in PyTorch [68]. We split the dataset into 90% training (36,000 samples), 7.5% validation (3,000 samples), and 2.5% testing (1,000 samples), and we use the Adam optimizer [54] with a learning rate of 1×10^{-3}, a learning rate exponential decay of 0.98 per epoch, and a batch size of 8. We train every model—the decoder, the encoder, and `direct-learning`—for 200 epochs. For our method and all baselines, we experiment with different regularizer weights

Table 9: Ablation study: linear encoder vs. non-linear encoder for soft-robot evaluated on test set. All encoders are trained on non-linear decoders with a regularizer weight of 0.05

	#successful cases (over 1,000)	Avg. distance to target on successful cases
Non-linear encoder	**975.0±3.9**	**0.0464±0.0018**
Linear encoder	710.7±24.8	0.1324±0.0276

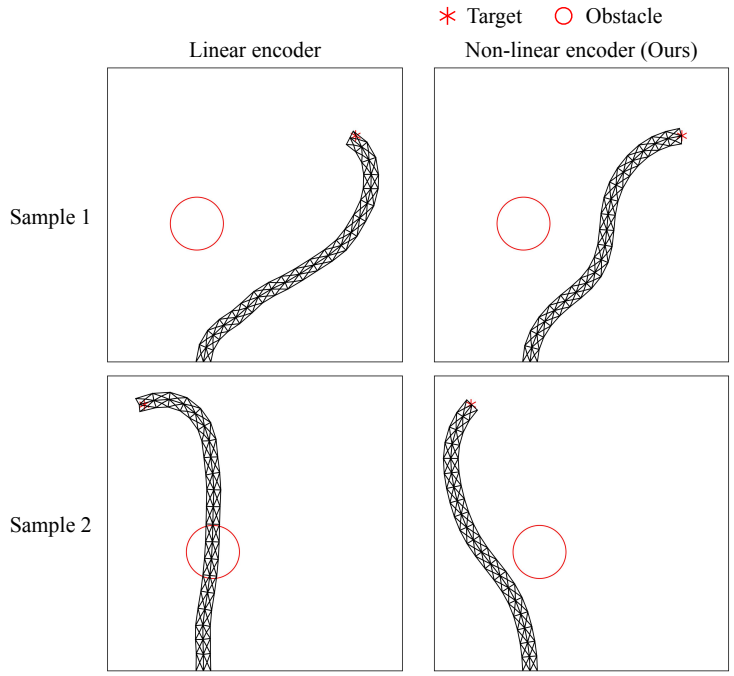


Figure 11: Soft-robot evaluation of linear encoder *vs.* non-linear encoder (ours) on examples from the test set. Linear encoder both violates the constraints (Sample 2) and misses the target (Samples 1 and 2).

$\lambda_2 = 0.03$, 0.05, 0.07, and 0.09. For `direct-optimization`, we use the BFGS implementation in SciPy [92], with a gradient tolerance of 1×10^{-7}. We use our internal cluster with 7 servers with 14 Intel(R) Xeon(R) CPUs. To train the needed neural networks, it takes approximately 2 hours for `direct-optimization`, approximately 4 hours for `direct-learning`, and approximately 4.5 hours for our method. Note that since we only need to train once, the training costs are amortized.

B.5 Failure cases

Since our encoder and decoder are both neural networks, there might be some generalization errors. We show two failure cases of our method in test set in Figure 10. The design proposed by our method misses the target by a short distance in the first example, and both touches the obstacle and misses the target by a short distance in the second example.

B.6 Ablation study: linear encoder

To demonstrate the non-linearity in our encoder is necessary, we train a linear encoder on the pre-trained non-linear decoder, following exactly the same training procedure mentioned in § B.4. We show the number of cases the robot successfully avoids the obstacle and the average Euclidean distance to the target for successful cases in Table 9. Linear encoder violates the obstacle constraints approximately 11.6 times as much as the non-linear encoder, and the average Euclidean distance for successful cases is approximately 2.9 times as much as the non-linear encoder. Two samples are shown in Figure 11. In both samples, the linear encoder misses the target by a short distance, and in the second sample, the linear encoder violates the obstacle constraint.