**Proceedings of the ASME 2023**
**International Design Engineering Technical Conferences and**
**Computers and Information in Engineering Conference**
**IDETC-CIE2023**
**August 20-23, 2023, Boston, Massachusetts**

**DETC2023-116678**



# WHEN IS IT ACTUALLY WORTH LEARNING INVERSE DESIGN?

**Milad Habibi[1], Jun Wang[2], Mark Fuge[1,*]**

[1]Center for Risk and Reliability, Department of Mechanical Engineering, University of Maryland, College Park, MD
[2]Department of Mechanical Engineering, Santa Clara University, Santa Clara, CA

## ABSTRACT

*Design optimization, and particularly adjoint-based multi-physics shape and topology optimization, is time-consuming and often requires expensive iterations to converge to desired designs. In response, researchers have developed Machine Learning (ML) approaches—often referred to as Inverse Design methods—to either replace or accelerate tools like Topology optimization (TO). However, these methods have their own hidden, non-trivial costs including that of data generation, training, and refinement of ML-produced designs. This begs the question: when is it actually worth learning Inverse Design, compared to just optimizing designs without ML assistance?*

*This paper quantitatively addresses this question by comparing the costs and benefits of three different Inverse Design ML model families on a Topology Optimization (TO) task, compared to just running the optimizer by itself. We explore the relationship between the size of training data and the predictive power of each ML model, as well as the computational and training costs of the models and the extent to which they accelerate or hinder TO convergence. The results demonstrate that simpler models, such as K-Nearest Neighbors and Random Forests, are more effective for TO warmstarting with limited training data, while more complex models, such as Deconvolutional Neural Networks, are preferable with more data. We also emphasize the need to balance the benefits of using larger training sets with the costs of data generation when selecting the appropriate ID model. Finally, the paper addresses some challenges that arise when using ML predictions to warmstart optimization, and provides some suggestions for budget and resource management.*

## 1. INTRODUCTION

Shape and Topology Optimization, or design optimization more generally, can be time-consuming and require expensive iterations to reach optimal results. For example, in Topology Optimization, solving forward problems requires an iterative algorithm to minimize objective functions typically via an additional adjoint-based backward pass to perform gradient-based optimization at each iteration. While these can be efficient for large-scale problems, existing gradient-based methods do still require multiple passes through an often computationally expensive simulator, and can get trapped in local optima, owing to their gradient-based nature. To overcome these challenges, researchers have studied methods that combine or supplement TO with inverse design and Machine learning methods. These methods can have significant time and resource savings compared to optimizing a design for each input condition, when compared with solving the forward model. For example, Chen *et al.* [15] showed that ML-based Inverse Design methods could predict an optimal airfoil shape to within 96% of the optimal efficiency compared to gradient-based methods, and more importantly, further warm-start Shape Optimization on the ML-predicted solution produced, on average, better solutions than gradient-based methods could achieve by themselves.

The use of ML-based design methods, however, come with an important, often unaddressed cost: one has to either find or generate data with which to train such models. For example, Woldseth *et al.* [34] investigated several previous studies in terms of the computational effort associated with generating training samples, running the learning algorithm, and applying the proposed procedure to obtain the optimal solution. They showed, for instance, Nakamura *et al.* [26] sampled 300,000 optimized structures for their training and validating data. They concluded that each Inverse Design model should be applied to at least 333,001 new similarly-sized problems to be "worth it" which they viewed as a high computational cost relative to just running a TO procedure [34].

This concern about the data needed for ML-based methods leads to an important, though often ignored question: when is it actually worth learning Inverse Design? That is, under what conditions would it actually make sense to go through the trouble of collecting the data for and then training any kind of ML method, as opposed to just expending the high computational cost needed to run a high-quality TO method for that same problem? This paper attempts to quantitatively address that question, by

*Corresponding author: fuge@umd.edu

specifically contributing the following:

1. We study how changing the amount of training data provided to several classes of Inverse Design models impacts both the Instantaneous and Cumulative Optimality Gaps of the generated designs on a specific 2D Heat Diffusion SIMP Topology Optimization problem. We show that, as expected, increasing training data size produces lower optimality gaps, and that the extent of this improvement is model and problem dependent.

2. We introduce several cost measures that quantify and differentiate different ID models and provide quantitative comparisons of those costs on an example ID problem. This allows us to broadly analyze the Return-on-Data-Investment (RODI) for a given model, taking into account costs of data generation, training, and warmstart optimization of the ML predicted solution.

3. We use these investment measures to quantify a "breakeven point" where Inverse Design produces positive return, compared to just running TO in isolation. We demonstrate how this breakeven point changes depending on the type of model and amount of training data used.

## 2. BACKGROUND AND RELATED WORK

### 2.1 Inverse Design

An inverse design problem belongs to the broader category of inverse problems. Inverse problems use observed measurements to infer model parameters [31]. The measurements can be obtained either by observing physical systems or by simulating them. The formulation for inverse design can be described as follow:

$$y = F(x) + \epsilon \tag{1}$$

where y $\in$ Y measurement data, x $\in$ X parameter of interests, F is the forward model which maps the parameter of interest to the measurement data, and $\epsilon$ is the noise of observed data. One of the main essential needs for using inverse problems is that the underlying governing equation on many applications are unknown or costly [4]. Therefore developing a surrogate model to map between observed data and parameters of interests are one of the main strategies for solving inverse problems.

Researchers have studied Inverse design in the field of mechanical engineering for a variety of problems including material design [3, 28], hydrodynamic design [32, 36], and aerodynamic shape design [15, 22].

Inverse design is also used extensively for warm starting optimization process [19, 20], as opposed to just using the ML-provided solution as-is. In warm start optimization, the ML-predicted solution is used as an initial point to reduce the running time of iterative methods. A common hypothesis is that if the initial ML-predicted solution is close to the true optimum, this warm start will significantly reduce the running time of traditional optimization [14, 17]. In comparison, this paper attempts to quantify such gains as a function of the amount of training data.

### 2.2 Topology optimization

Topology optimization (TO) determines how to place material in a design domain to optimize one or more objectives while satisfying constraints [29]. One of the most widely used formulations of TO problems are pseudo-density-based approaches such as the Solid Isotropic Material with Penalization method (SIMP) [5]. Both Topology and Shape Optimization are widely used for finding local and global optima for a variety of design tasks [5, 6]. Gradient-based topology optimization has been widely studied to place material within a design domain. However, computing the solutions to such optimization tasks can be computationally prohibitive, motivating the use of fast, Machine Learning based approximations of the optimal solution [13, 21, 24, 33]. Recently, promising work in applying ML to Optimization problems has focused on Deep Generative Models [16, 27], and specifically, the incorporation of physics or other engineering constraints into those models [23]. Beyond direct Inverse Design prediction as a supervised learning problem, some research has alternatively formulated this as a reinforcement learning problem [10].

The above works demonstrate a variety of methods for integrating deep learning into design optimization. However, all are either supervised or unsupervised algorithms that rely on the use of training data to learn the ML model. In contrast to this paper, none rigorously or quantitatively address the important question of when such models are worth the return on time or cost needed to generate the training data or how the performance improves or degrades as we change that cost.

### 2.3 Return-on-Data-Investment

While the above-mentioned papers have shown that using ML-based methods for Inverse Design can be possible and performant, they come with a non-trivial cost: the need to generate training data to feed into the model. To understand under what conditions generating this data is "worth it," we need to define a way to measure the impact on a model's performance as we generate additional data. We refer to this throughout the paper as the Return-on-Data-Investment (RODI), after common Return on Investment (ROI) measures used in economics.

Finding the ROI parameter plays a key role in decision-making in engineering and business [11]. As we use more complex ML models to improve ID results, the data investment costs often increase nonlinearly. There is, however, a point at which the quality and performance of a given ML model saturates, implying that there is no benefit to adding more complexity or data to the models [18]. For instance, Deshpande *et al.* [18], analyzed Random Forest and Bidirectional Encoder Representations from Transformers, based on the accuracy and ROI for two publicly available data sets and they recommend selecting ML classification not solely based on performance, but also given how much data one has available.

Calculating a Return on data investment is one of the most important parameters that can help us address when Inverse Design is actually worth it. In the TO community, this motivation has not received much attention and there are few studies on it. For example, Woldseth *et al.* [34], introduced the computational costs in general terms for ML-based TO including the actual solu-

tion time, collecting data, and quality needed for the performance. They compared some methods in the reviewed literature via the perceived generalization ability to when speed-up is achieved, and defined a qualitative breakeven threshold. To make the analysis of that breakeven point more precise, one of this paper's main contributions is to study the exact relationship between three driving costs on a concrete common example: (1) TO iterations, which also drives Data Generation Cost; (2) ML model choice, which drives Model Training Cost; and (3) a model's ID performance quality, which drives its Relative Performance Improvement (defined below) with respect to TO. We compare these across a range of models and on common problems such that we can compare them fairly. The next section further details our test problem and exact cost measures.

## 3. METHODOLOGY

To address the contributions mentioned in the introduction, our methodology is divided into the following sections: (1) Defining our Heat condition topology optimization problem and how we generate the data set, (2) how we train and optimize our models, and (3) how we measure and evaluate the performance and cost results.

### 3.1 Heat conduction topology optimization problem

While research literature uses a variety of TO test problems, we chose to use a fast-to-evaluate and simple to replicate 2D Heat Conduction test problem that might serve as a useful baseline (§5 addresses other or more complex problems). Topology optimization has been successfully applied to determine the material distribution in a variety of heat conduction applications [25], and the purpose of these studies was to find the optimal material distribution for a heat transfer problem inside a region of interest while satisfying design constraints. For instance, Yang *et al.* investigated the optimality of structures optimization of heat conduction structures for minimum thermal compliance while satisfying minimum-maximum temperature conditions [35].

In this study, we consider a 2D heat sink problem subjected to pure conduction. This problem can be described as finding the material distribution that minimizes the integral of the temperature when the amount of highly conducting material is limited. This Test case was derived from a demo example described in the dofin-adjoint solver [1] and which we had used in a prior study of Inverse Design methods [8]. The goal of this problem is to minimize the thermal compliance subjected to the Poisson equation with mixed Dirichlet–Neumann conditions [1], which formulate as follows:

$$\int_\Omega fT + \alpha \int_\Omega \nabla a \cdot \nabla a \tag{2}$$

where f is the heat source term (here is a constant $10^{-2}$), T is the temperature, $\Omega$ is the region of interest (unit square), $\alpha$ is a regularization term, and a is the mass distribution function (a(x) = 1 for material, a(x) = 0 for no material). It is subjected to a control constraint over the domain:

$$\int_\Omega a \leq V \tag{3}$$

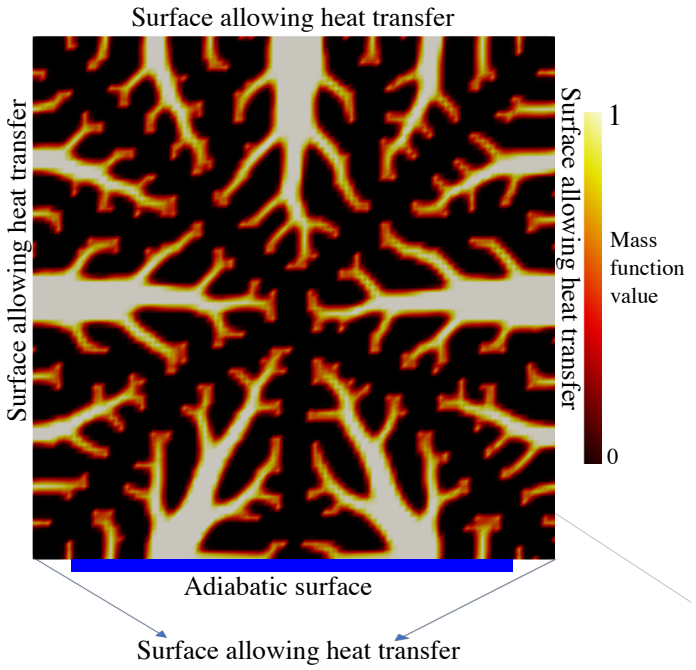


**FIGURE 1: THE PHYSICAL LAYOUT OF THE TOPOLOGY OPTIMIZATION PROBLEM WE USED IN THIS PAPER. THE COLOR BAR REPRESENTS THE VALUE OF THE MASS FUNCTION AT ANY POINT.**

$$a \in [0, 1] \tag{4}$$

Where $V$ is the volume bound over the region of the interest domain $\Omega$.

**3.1.1 Dataset and Preprocessing.** In this topology optimization problem, we aim to minimize the compliance of the domain upon a given limit on the volume of conducting material and length of the adiabatic region as design parameters similar to our previous study [8]. The adiabatic region is shown with a blue color in Fig 1.

To generate the data set needed for this study, we used two input parameters (design parameters): the volume limits on the material distributions and the length of the adiabatic region. The physical layout of this topology optimization problem is shown in Fig 1. We chose adiabatic length which is shown with a blue region at the bottom of the geometry between 0 and 1 and bounded the upper volume limit between 0.3 and 0.6, since our interior solver (IPOPT) cannot produce converged results outside of these ranges. Each of these parameters was divided into 20 equal segments, which resulted in 21 values for each. Hence, using 21*21 design parameters, 421 different optimized designs were generated.

In order to show the details without increasing the computational time, a mesh 100 x 100 was used as the design domain. In addition, every combination of design parameters was run for 100 iterations until IPOPT satisfied the tolerance of 1.0e-100. For every combination design space, five values are collected for each point, including x-coordinate, y-coordinate, volume bound, adiabatic region length, and mass function value.

### 3.2 Machine Learning Models Compared in this Paper

Understanding under what conditions data collection and model training is actually worthwhile for Inverse Design clearly depends on the quality and sample efficiency of the underlying Machine Learning model one might use. While there are a large number of possible models, we selected a representative subset of supervised learning models spanning a range of complexities to compare in this paper. These range across three main model families of non-linear Supervised Learning approaches from fairly simple prototype-based methods, such as K-Nearest Neighbors (KNN), to Ensembles such as Random Forests, and to Adaptive Basis Function methods, such as Deconvolutional Neural Networks (DeCNN). This section briefly reviews each of these models and provides citations to further reading for interested readers.

**K-Nearest Neighbors:** K-nearest neighbors (KNN) is a non-parametric and supervised learning algorithm used for classification and regression. We used a regression model of KNN, whose predictions for a given test point are weighted averages of the k-closest training data points. This algorithm uses different types of distances to compute that proximity, such as Euclidean distance, Manhattan distance, and Minkowski distance, and can modify the number of neighbors (k) over which it takes a weighted average [30]. We chose to use a KNN model for comparison, since it is straightforward to implement, fast to evaluate for a small number of samples, has a limited range of hyper-parameters requiring tuning, and has fast and deterministic training procedures. In this sense, while it is not as capable of generalization compared to other models, it represents a model with comparatively "low-cost."

**Random Forests**: A Random Forest (RF) is an ensemble learning method for classification and regression which builds a number of decision trees using independent bootstrapping samples of the dataset [9]. We used the RF regression model which predicts a given test point based on averaging over multiple decision trees employed over training data sets. In this algorithm, trees are run in parallel, meaning there is no interaction between them. We used an RF model for comparison since it is one of the most widely used ensemble learning models, while also possessing a limited number of hyperparameters that require tuning, thus limiting required computational costs.

**Deconvolutional Neural Network**: Deconvolutional neural networks (DeCNNs) are also called transposed convolution neural networks, which use a special case of convolution to perform weight-sharing within a feedforward network. At the time of writing, Deconvolutional Neural Network architectures were among the most commonly used by contemporary papers attempting to do Inverse Design of Topologically or Shape Optimized problems, and thus represent a natural comparison case [37]. Their popularity is due to the comparatively large hypothesis class of functions that DeCNNs can learn, owing to their large model capacity. This has led to generally lower test Mean Squared Error on Inverse Design tasks compared to simpler models, although DeCNN's possess their own shortcomings. For example, they have many possible hyper-parameters that need to be optimized, such as training epochs, learning rates, regularization strengths, and neural network architectures, making them harder to train and optimize with stable variance given a small number of training samples. In addition, the non-convexity of the training loss surface often necessitates additional diagnostic checks or multiple restarts during training, further increasing the cost and complexity of training such models. A common criticism of such DeCNN models in Inverse Design is that, while their performance may appear to be relatively strong, their needed training time and complexity, along with their dependence on a larger training sample size may not make them "worth it" [34]. This paper helps illuminate under what conditions that conjecture might be true.

**3.2.1 Model Training, Hyperparameter Optimization, and Data Postprocessing.** To conduct our subsequent experiments, we had to perform several steps for creating a cross-validation set, optimizing each model's hyperparameters, downsampling the training datasets, and pre-processing the input data to make it compatible with each model.

**Cross Validation and Hyperparameter Optimization**: To perform cross-validation and hyperparameter optimization, we randomly selected two unique values of adiabatic length and volume limits from the dataset and excluded them from training data to act as a test set. After this, we randomly selected half of the excluded data as validation data and the other half as test data. To find the point-wise mean squared error for hyperparameter optimization, we tested each models' predictive abilities on validation data points corresponding to the topologies defined in the excluded validation dataset.

We used the KNN and RF implementations from the scikit-learn library [12]. In the KNN model, hyperparameter optimization involves both the weighting and number of neighbors. In the RF model, we optimize the number of estimators as well as the minimum number of samples in newly created leaves. We implemented the DeCNN model using Tensorflow [2] and optimized the learning rate based on the maximum batch sizes possible for each training size.

All models were trained on the different training datasets described above, and each model's hyperparameters were optimized on the common validation data. Therefore, hyperparameters are chosen for each model such that it produced the best performance, as measured by point-wise mean square error (PMSE) on a validation dataset. We exclude the test data from both the training and validation datasets, such that the test errors are not influenced by our hyperparameter selection approaches.

**Training Dataset Downsampling**: To study how each model performs when trained on varying amounts of data, we gradually reduce the size of the training dataset. After excluding the validation and test dataset—which we will keep common across all models—the largest training data set includes 361 designs. Every subsequent training set is obtained by randomly removing data from the next largest set. For example, the training dataset with a size of 200 is obtained by removing 50 data from the training dataset with a size of 250. In this way, the 200 size dataset is a strict subset of the 250 size dataset. This ensures that any difference in performance between models of different sizes is due only to additional training points. Below, we evaluate the models on training data sets ranging in size from 2 to 361 designs.

**Data Postprocessing**: During our model evaluation process,

the thermal compliance (the objective value) and trajectory are normalized and averaged. The specific postprocessing procedure is:

- The objective value of each model/data size is normalized with respect to the optimal value obtained in the control trajectory (*i.e.*, a uniform initialization with no ML warm-starting).
- The number of iterations of each trajectory extended to the maximum number of iterations if the run that converged prior to the maximum. Taking the optimal value that reached the trajectory and extrapolating it to the remaining iterations, makes it easier to visually compare different trajectories.
- The average objective values and the 95% Empirical CI on the average percentile are calculated for each trajectory and shown as a solid line and shaded color, respectively.

**3.2.2 Return in Data Investment Measures.** To define an actual breakeven point where actually ML-based Inverse Design is worth it, this section defines the overall computational costs of generating training samples, computational effort associated with running the learning model, and the desired performance quality. These cost measures allow us to talk about trade offs among them. Our first step is to introduce the costs associated with iterative solvers and training models. Then we describe the relative percentage performance improvement, which is essentially a measure of solution quality. Understanding return on data investment requires describing trade-offs among all the above-mentioned parameters.

**Data Generation Cost**: The computational cost of ML-based inverse design is not only related to the machine learning model, but also time spent on generating sample training data. Creating training samples is usually expensive and depends on the type of solvers used, the quality needed (such as the resolution or fidelity) and convergence criteria, as well as the computational resources used to prepare the data. In this study, we fix the quality needed for training data samples such as resolution and convergence criteria as described in Sec. 3.1.1 which quantifies the number of iterations needed for the solver to converge the desired criteria. To establish a common basis among different examples, this cost is converted into the time needed for simulating each problem on an Intel(R) E5-1620 CPU. Herein, we convert the generation data cost to time spent on the mentioned CPU for this generation. Later sections will discuss how hardware changes may affect our results.

**Model training cost:** The model training cost plays a key role in choosing among ML models. We used various ML models, each of which has different computational costs for training. We measure this cost via training each model on a single CPU E5-1620 Intel(R) similar to data generation costs. Therefore, for various amounts of training data and ML-models we report the training cost of each model with time values measured on a single CPU. Later sections will discuss how hardware changes may affect our results.

**Relative performance improvement**: The other important factor in calculating our return data investment is how well the trained model produces high quality (*i.e.*, high performing) solutions. We will use the relative percentage improvement (RPI) in each problem's objective function to measure this, formulated as follows:

$$RPI = \frac{C^{Init}_{CONTROL} - C^{ML}}{C^{Init}_{CONTROL} - C^{OPT}_{CONTROL}} \times 100\% \tag{5}$$

where $C^{Init}_{CONTROL}$ is initial compliance (first iteration) calculated based on a uniform initialization of the SIMP method, $C^{OPT}_{CONTROL}$ is optimal compliance (last iteration) calculated based on the final converged SIMP method, $C^{ML}$ is the compliance of the design computed based on ML models prediction. Therefore, the return on data investment can be interpreted as a tradeoff between the RPI and the data generation and model training costs.

## 4. EXPERIMENTAL RESULTS AND DISCUSSION

Given this methodology, below we first demonstrate the impact of different training sizes for different ID models on the warm start initialization performance. The shaded parts of these plots represent 95% confidence intervals for corresponding plotted functions. Following that, we plot the return on data investment to determine when different ID methods can find designs with lower Instantaneous and Cumulative Optimality Gaps as a function of additional training data samples. As we detail further below, this curve, coupled with the cost associated with generating additional training data, helps us determine the return on data investment.

### 4.1 How well do ID models warmstart TO?

**KNN**: using hyperparameters optimized for each size of training data, we now compare how well a simple KNN model can predict the optimal geometry, as well as that prediction's usefulness as a warm start for further topology optimization. As a baseline, we compare it with uniform initialization commonly used in TO (We label this baseline "Control").

On average, we found that a KNN model with all sizes of training data from 15 to 361 samples outperformed the control condition. Furthermore, when the size of training data exceeds two samples, the KNN models produced predictions with thermal compliance values that were significantly lower than the control model. As well as improving prediction compliance, we found that increasing the number of training sizes accelerates convergence to optimal compliance value. Figure 2 plots a subset of the training data sizes for clarity's sake, and plots containing the full set of results are located in the Supplemental Material. As Fig. 2 shows, in all warm start trajectories the optimizer increases the thermal compliance in early iterations of warm starting (see around iteration 7), and then it accelerates the convergence to optimal compliance values compared to the control condition. These early "spikes" in the objective function value (roughly between iterations two to ten) are related to the IPOPT solver, and we detail this behavior in the later discussion section.

**RF**: we use hyperparameters optimized for each size of training data to predict the optimal geometries for test data in order to compare the effects of changing training size on RF predictions.

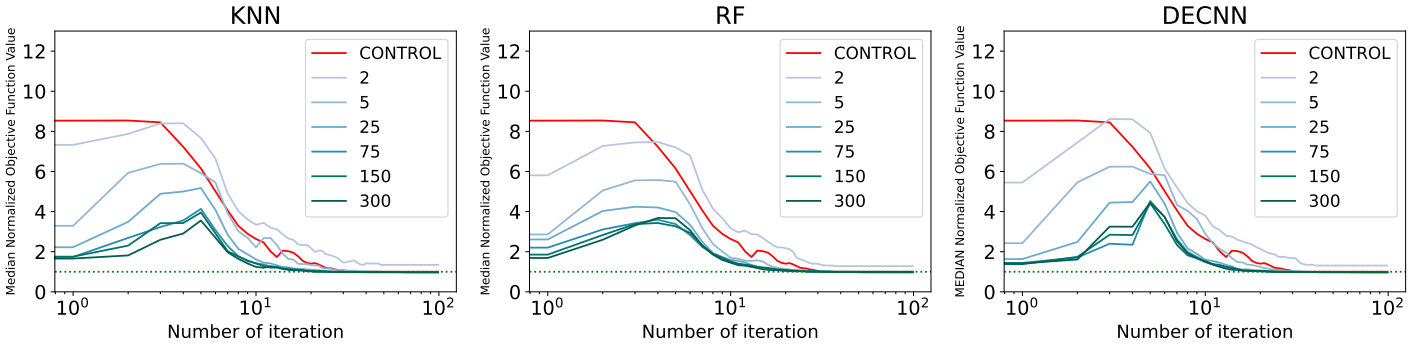


**FIGURE 2: 2D HEAT CONDUCTION OPTIMIZATION TRAJECTORIES WARMSTARTED WITH ALL THREE ID MODELS, TRAINED ON THE NUMBER OF DATA NOTED IN THE LEGEND. THE 'CONTROL' TRAJECTORY IS INITIALIZED WITH A CONSTANT DISTRIBUTION SET TO THE VOLUME FRACTION. TRAJECTORIES OF FURTHER DATA SIZES CAN BE FOUND IN THE SUPPLEMENTAL MATERIAL.**

Then, we compare the performance of each training size prediction as optimal geometry and a warm start for further topology optimization.

On average, Fig. 2 shows that the RF model with training set size of more than seventy five converges faster to the optimal design compared to the control condition. Fig. 2 shows that the same increasing trend in compliance occurs in the early stages of RF warm start trajectories as it does for KNN trajectories. In addition, the performance of RF models improves as training sizes are increased.

**DeCNN**: we use the hyperparameter optimized for each size of training data to train the deconvolutional neural network model. We found that on average the DeCNN model predicts geometries with lower initial optimality gap compared to the RF and KNN models. As above, we compare the performance of different training sizes at predicting the optimal geometry and as a warm start for further topology optimization. Figure 2 provides a subset of training sizes, with the full version provided in the Appendix. Figure 2 shows that the DeCNN model with training sizes beyond twenty five design samples outperforms the control condition. Also, the DeCNN model can predict optimal designs with significantly lower thermal compliance with respect to the control condition. In the early stages of the warm start trajectory, compliance increases similarly to the KNN and RF models.

Overall, our results in Fig. 2 show that, with a limited amount of training data ($\approx$ 5+), the KNN model is the most effective in terms of convergence speed. However, when the amount of training data increases ($\approx$ 25+), the DeCNN model outperforms the other models in terms of both the initial design prediction as well as convergence acceleration, although simpler models like KNN and RF also improve with increased training data. Our findings suggest (perhaps somewhat expectedly) that when warmstarting TO with limited training sizes, simpler models can be more effective, whereas with more data more complex models like DeCNN produce better warmstarts.

### 4.2 How well do ID models predict the optimal design without warmstarting?

What if we do not have access to a TO solver to warmstart further optimization and wish instead to just use directly the geometry or design output by the ID model?

Figure 3 relates the size of training data to the Normalized Initial Optimality Gap, which measures how close in performance to the Control (TO) solution the initial ID model's prediction gets before further optimization. Figure 3 shows the initial optimally gap reduces, on average, when we increase the amount of training data.

As expected, we observe that increasing the amount of training data decreases the initial optimally gap, that is, more data improves the ML models' prediction accuracy. In this measure, the DeCNN model outperforms the other models, on average. This suggests that if we only care about an ID model's prediction without further optimization, the DeCNN model is the best choice among the models studied in this paper. Notably, none of the models ever achieved complete parity in performance with the Control (TO-only) solution.

### 4.3 How much data do ID models actually need to accelerate optimization?

The whole point of attempting Inverse Design is to reduce needed optimization effort on subsequent design cases. As such, one of the most important costs we need to track is how increasing the data budget reduces the computational budget we need to spend on further optimization of new or unseen problems. If investing in an ID method significantly reduces subsequent optimization costs, then perhaps the return on data and model investment is worth it; otherwise, it may not be.

To show the effect of data set investment versus further optimization cost, we measured the relative performance improvement of three machine learning models versus the time cost of warm start optimization—that is, how much further time did we need to spend to optimize the ID provided solution to achieve results similar to TO without the ML-based warmstart? Figure 4 varies the amount of training data given to each model.

Expectedly, Fig. 4 depicts that as each ID model is given a larger training size, each outputs designs that have better Relative Performance Improvement (Eqn. 5), *i.e.*, are closer to the optimal thermal compliance found by the TO-only Control solution. Among the models, the DeCNN outperformed the other two models in terms of initial relative performance. The DeCNN model with 300 training data produces the best initial relative performance improvement, whereas the KNN model with 5 training

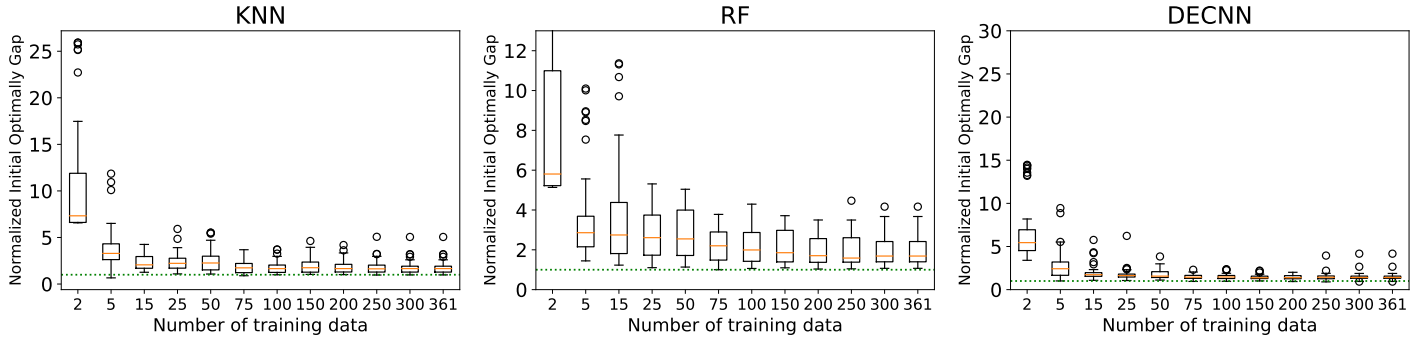


**FIGURE 3: THE NORMALIZED INITIAL OPTIMALITY GAP OF THE KNN, RF, AND DECNN MODELS AS A FUNCTION OF TRAINING DATA SIZE. THE ORANGE LINES REPRESENT IS THE MEDIAN OF EACH BOX-PLOT AND THE DASHED GREEN LINE REPRESENTS THE OPTIMAL VALUE ACHIEVED UNDER UNIFORM INITIALIZATION (THE CONTROL CONDITION).**

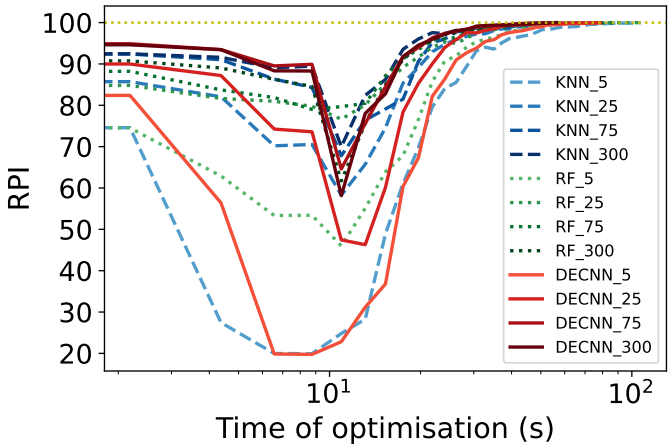


**FIGURE 4: RPI TRAJECTORIES USING WARMSTART OPTIMIZATION, WHERE THE BLUE/GREEN/RED CURVES CORRESPOND TO THE KNN/RF/DECNN MODELS, RESPECTIVELY, AT THE NOTED TRAINING DATA SIZE.**

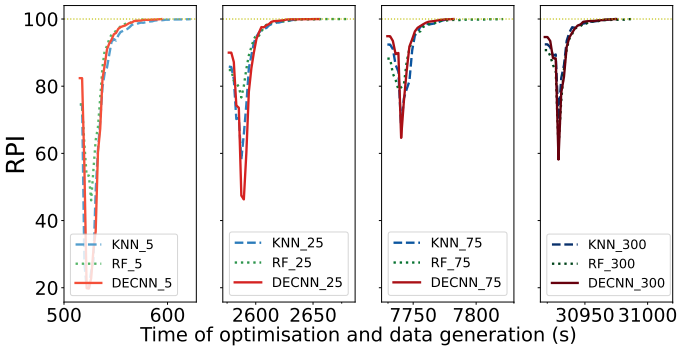


**FIGURE 5: COMPARISON BETWEEN OF THE TIME SPENT ON OPTIMIZATION AND DATA GENERATION FOR WARMSTARTED OPTIMIZATION TRAJECTORIES WITH VARIOUS MODELS AND TRAINING DATA SIZES.**

data shows the lowest initial relative performance improvement. Furthermore, when used to warm start further optimization, the DeCNN trained with 300 training data requires the fewest iterations to converge to the optima, compared to the control solution. This implies that, when given at least some small amount of training data, ID methods can meaningfully accelerate TO convergence.

### 4.4 What about the cost of data generation?

While our earlier results would seem to imply that a DeCNN with 300 training samples is best, this ignores both the data generation cost and training costs. To capture data generation cost, Fig. 5 compares the time spent on both warmstart optimization and data generation. Factoring in data generation, the total cost increases as the amount of training data grows—note in particular the different x-axis scales. As expected, the models that predict designs with the best initial RPI also have the highest training data costs.

Figure 5 suggests that the amount of data used to train ID models has a significant positive impact on their performance. Past literature also uses large dataset sizes to train ID models, such as in [26] where the authors create 300,000 optimized structures for their training and validation data. As expected, using larger training sets leads to better ID-predicted designs and improved relative performance improvement. But at what cost? Existing papers often ignore the non-trivial sunk cost of generating this data. Specifically, the models that produced the best initial relative performance improvement also had the highest data generation costs, implying a trade off between the compute cost we wish to "invest" into data generation versus the amount the trained ID model would "save" on downstream optimization. For example, while Fig. 4 showed that a DeCNN model trained with a large number of samples accelerated optimization the fastest, when we include the time needed to generate training data in Fig. 5 the hidden time cost of this data generation becomes plain to see.

### 4.5 What about the cost of training time?

From Fig. 5 alone, it would again appear that subject to similar training data amounts, a DeCNN model outperforms others; however this ignores the model training cost/time. Figure 6 provides a comparison of various inverse design optimization trajectories that were warm-started while adding in the amount of training time required for each model. (Note: in Fig. 6 we

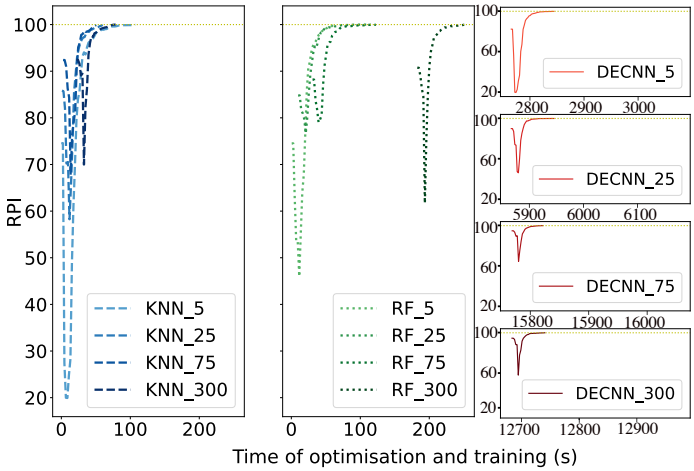


**FIGURE 6: COMPARISON BETWEEN OF THE TIME SPENT ON OPTIMIZATION AND MODEL TRAINING FOR WARM STARTED OPTIMIZATION TRAJECTORIES WITH VARIOUS MODELS AND TRAINING DATA SIZES.**

have removed the data generation costs of Fig. 5 so as to only demonstrate the differences in training time.) The results illustrate that the DeCNN has the most significant training cost when compared to other models and that cost depends strongly on the training data amount.

Several studies have emphasized the importance of training cost in model selection. For instance, Bengio *et al.* found that the cost of training deep neural networks is a significant barrier to the adoption of deep learning in the industry [7]. Figure 6 demonstrates similar effects in ID models, where we observe that the DeCNN model outperforms other models in terms of initial relative performance, but it has far higher training cost compared to inexpensive models such as KNN and RF (for simplicity, the figure ignores the data generation costs show in Fig. 5). In fairness, the time costs here were all normalized to CPU time for consistency across the paper, and in practice one could take greater care to leverage distributed training across GPU resources as we describe later, but even under those cases the training costs would remain non-trivial. We suspect this result is painfully obvious to any readers who have actually tried training a DeCNN model on these types of problems, but hopefully it is illustrative to readers who have not.

### 4.6 What if I had different cost tradeoffs between data generation, optimization, and training?

Our above comparisons are limited to our chosen example problem and, more specifically, the fact that the optimization, data generation, and training time costs are all treated equally as normalized to common CPU time for our particular computing platform. How would things change if the relative costs of optimization, data-generation, and training differed? For example, if we already had an existing dataset, or if we could parallelize the training costs? To consider cases where these relative costs change, Fig. 7 compares different relative weights between model training, data generation, and warmstart optimization (*i.e.*, running further TO on the ID model provided solution).

In that figure, we simplify each model's optimization trajectory by creating a line connecting the initial design predicted by the ID method with the corresponding design after completing warmstart optimization—as such, these end points will generally have an RPI of 100% unless the warmstart traps the TO solver in a local optima. The total time cost on the x-axis is calculated based on a weighted average of the three costs described above: optimization cost (the cost of warmstart iterations), data generation cost, and model training cost. The simplex on the left-hand side of Figure 7 allows us to define different weights for each time cost using Barycentric coordinates, and the right-hand side compares different models under the corresponding cost weight.

Figure 7.A describes the situation we have used thus far throughout the paper where the model training, data generation, and optimization costs have equal weight. As we saw above, in this case the KNN and RF models with lower training data size outperform the more complex DeCNN model, largely due to their lower model training cost and the relative inexpense of running additional TO optimization iteratons for this simple 2D thermal compliance problem.

Figure 7.B describes the situation where data generation is effectively free, while the model training and optimization costs have equal weight. In practice, this could occur when an existing dataset is available, but a researcher or company would need to bear the burden of model training and further optimization. In such cases, a KNN with a larger training size outperforms other models due largely to its lower training cost.

Figure 7.C describes the situation where we ignore the model training cost (*i.e.*, training is free), and optimization and data generation have equal weight. In practice, this could occur when training is hardware parallelized or where the training cost is borne by a third party. In such cases, complex models such as the DeCNN with smaller amounts of training data produce better return compared to simple models such as a KNN.

Figure 7.D describes the situation where we heavily discount the costs of both data generation and model training, compared to warmstart optimization. In practice, this could occur in cases where we use high quality pre-trained models that are fine-tuned on small set of domain-specific training samples, but where the optimization task itself is extremely costly—such as a 3D multiphysics optimization problem with multiple constraints. This is the most common setting in "Transfer Learning" approaches to Inverse Design. In such cases, Fig. 7.D suggests that even simple models trained on small-to-moderate amounts of training data may produce greater return than investing in larger training datasets or complex models.

Note the large difference between point D in Figure 7 compared to the nearby edge of the simplex where both data generation and model training costs are considered "free"—the earlier Fig. 4 showed this exact case where complex DeCNN models with large amounts of training data appeared to dominate all other models.

### 4.7 When do ID methods break even compared to just using TO?

One of this paper's main objectives is to determine the break-even point where a machine learning (ML) based inverse design method generates positive returns on investment, compared to

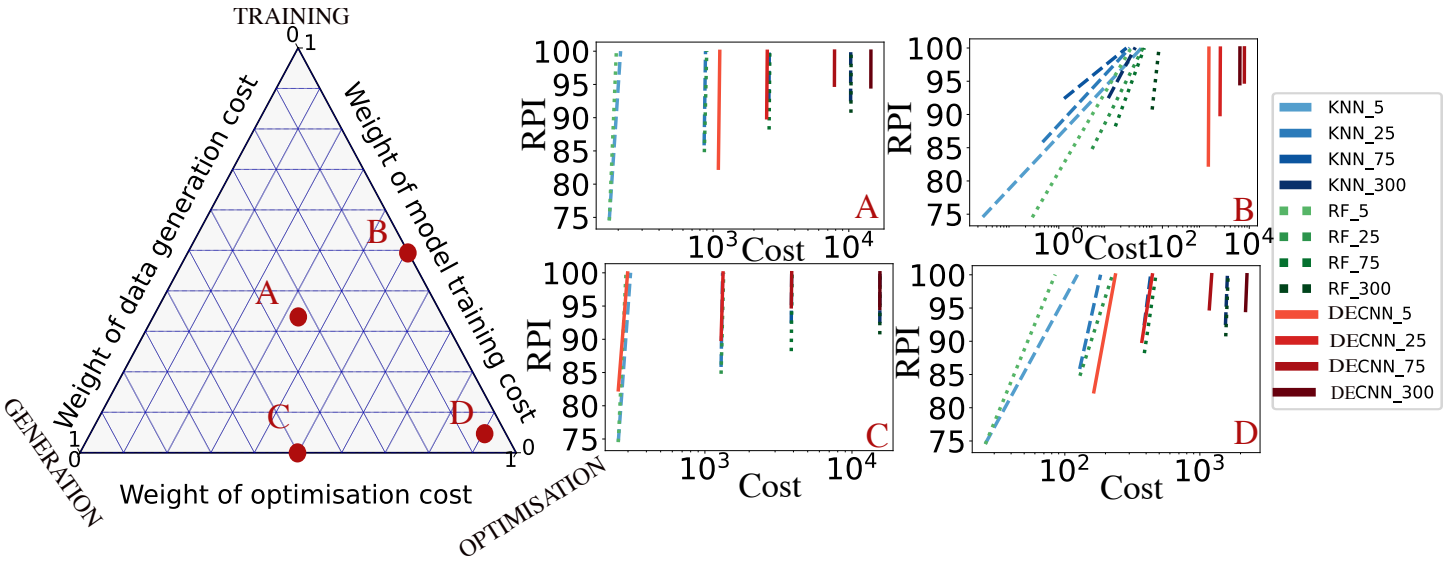


**FIGURE 7: LEFT: SIMPLEX DIAGRAM OF RELATIVE WEIGHTS BETWEEN DATA GENERATION, OPTIMIZATION, AND MODEL TRAINING COSTS. RIGHT: COST VERSUS RPI FOR DIFFERENT MODELS AND TRAINING DATA SIZES. A: EQUAL WEIGHT. B: DATA GENERATION HAS NO COST AND MODEL TRAINING AND OPTIMIZATION HAVE THE SAME WEIGHT. C: MODEL TRAINING HAS NO COST AND DATA GENERATION AND OPTIMIZATION HAVE THE SAME WEIGHT. D: OPTIMIZATION HAS 20 TIMES THE WEIGHT OF DATA GENERATION AND TRAINING.**

only using topology optimization. Up until now our focus has been solely on comparing the cost of generating a single predicted solution. This is an unfair comparison. In practice, no sane person would invest in data generation and model training unless they were able to *amortize* that fixed cost over many future design tasks to reduce their variable costs.

To analyze this trade-off between an upfront fixed investment and a reduction in variable costs, we first calculated the median number of warmstart optimization iterations each model required—this represents the variable cost of using a model to predict and then optimize a new design. To compute the fixed costs, we added each model's data generation cost and training cost. For comparison, we calculated the median number of iterations and corresponding costs for the control method (TO-only), which represents the variable costs of the TO solver, and we note that the control condition has no fixed cost, since there is no data or training.

Figure 8 plots both the fixed and variable costs of the control condition and the various models under different amounts of training data cost. Our results demonstrate that after a certain breakeven threshold of the number of new designs, some ML-based inverse design models outperform the control in terms of cost. For example, in the top left diagram, if you "invest" in generating 5 training data points, then an RF model would be less costly than the Control condition if you needed to compute 47 or more new designs. In the top right figure, after investing in 25 training data points, a KNN model breaks even with the Control condition after 168 designs. Furthermore, the bottom left diagram demonstrates that investing in 75 training points leads the DeCNN and KNN models to break even with Control after 1074 and 294 new designs, respectively. Lastly, the bottom right diagram shows that DeCNN and KNN models require you to use the model for 2492 and 1768 new designs, respectively, to recoup the 300 training data investment. Interestingly, even though complex models trained on large datasets have faster TO convergence—seen as a lower slope and thus variable cost in Fig. 8—they have larger breakeven points, owing to the fixed costs required for data generation and training.

## 5. DISCUSSION AND LIMITATIONS

### 5.1 When do ML models provide suboptimal warmstart conditions?

One challenge we observed is that in a small subset of cases an ML model's warmstart initialization was suboptimal compared to uniform initialization, leading the IPOPT solver to get stuck in a local optima. This resulted in a small handful of cases that highly skewed the mean optimization trajectories for each model, which is why we chose to report each model's median value for the optimization trajectories since it is less sensitive to outliers. Nevertheless, it is important to discuss here the fact that in a small percentage of cases ID methods might produce warm start initializations that are worst than uniform initialization.

### 5.2 Effect of hardware parallelism

Using hardware parallelism, GPUs can significantly improve both the training and hyper-parameter optimization time of machine learning models. While studying parallelization was not a rigorous focus of this paper, anecdotally, we observed that using GPUs could reduce the DeCNN training costs by about one order of magnitude. This case can be considered similar to Fig. 7.C in which we assume that model training is hardware parallelized, and as a result, the DeCNN model with smaller amounts of training data performs better compared to simple models such as the KNN. This suggests that parallelization plays a crucial role in making complex models more viable for inverse design and improves the return on investment in such models.

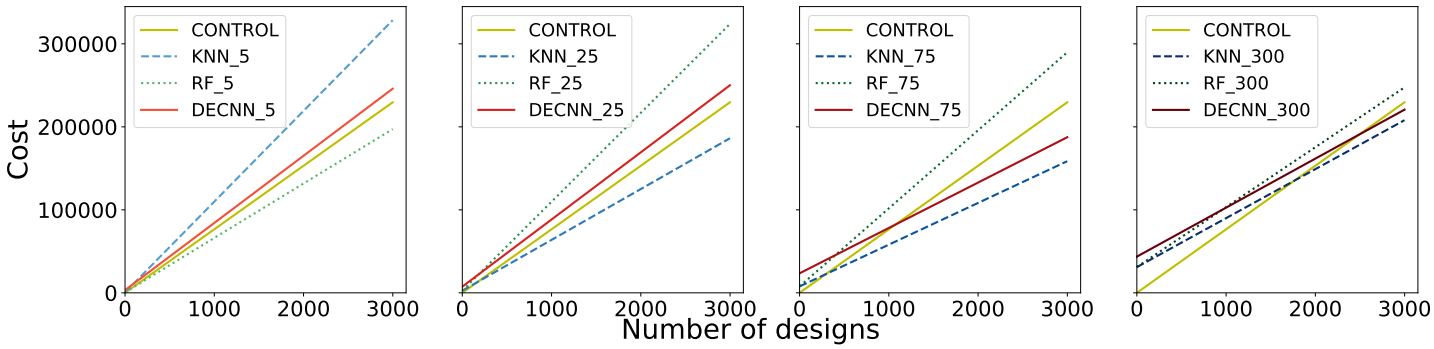


**FIGURE 8: THE COMPARISON BETWEEN THE CONTROL AND KNN/RF/DECNN COSTS, RELATIVE TO THE NUMBER OF NEW DESIGNS TO BE OPTIMIZED, FOR DIFFERENT TRAINING DATA SIZES—5, 25, 75, & 300 DATA POINTS.**

Similarly, in Fig. 7.D, where both data generation and model training costs are heavily discounted compared to warm start optimization, it is also possible that parallelism in terms of data generation (for example, on a compute cluster) could have a significant impact on the results.

### 5.3 Why does the thermal compliance increase after warm-starting?

In our previous paper, we presented a visual demonstration of designs that exhibit an early increase in compliance when using warm-starting [8]. To gain insight into the cause of this behavior, we conducted a comparison of the IPOPT and scipy solvers using the sequential least squares programming (SLSQP) method which the results are located in the Supplemental Material. We observed the scipy solver exhibits a gradual, monotonic decrease, while the trajectory of the warm start optimization with IPOPT displays an initial increase in compliance. We hypothesize that IPOPT behaves this way due to its solution method, which numerically approximates the hessian matrix during the initial solver iterations, and thus may take early sub-optimal steps. We hypothesize that this explains initial "bump" in the convergence trajectory.

### 5.4 Limitations in the test problem

This paper focused on a 2D heat diffusion problem with a limited volume fraction range of 0.3 to 0.6 and an adiabatic region width that can vary from 0.0 to 1.0. While this choice helped us obtain useful insights into this problem, it is natural to ask whether our results would generalize to three dimensional or more complex problems. Future work could explore the extent to which each model's cost-benefit behavior changes under increased complexity. Additionally, to assess the transferability of our claims to other problems, it would be beneficial to compare our results across multiple Inverse Design problems.

Relatedly, this paper did not consider other optimizers beyond the IPOPT solver, and less-efficient TO solvers will shift the costs-benefit trade-off substantially by increasing the costs of both the optimizer and the data-generation reducing the comparative effect of training costs. There are also additional hidden costs such as hyperparameter optimization of ML models, which can add significantly to training-time costs. We did not consider those rigorously here for sake of compactness and brevity, but future work can investigate the impact of these hidden costs on the performance of the model and explore alternative optimization strategies to address them.

## 6. CONCLUSION

So, when is it actually worth learning Inverse Design? Our answer ranges from "Almost Never" to "Worth Experimenting With" to "Almost Always" depending on the design situation.

**Almost Never** If the only performance metric we care about is the ability of an ID model to accurately and reliably predict close to the optimal design without further warmstarting, then §4.2 and Fig. 7 imply that only a complex model (such as the DeCNN or better) trained on large quantities of training data (75+ in our example), can achieve an RPI even close to 95% when compared to a TO-only control condition. Such models incur large training and data generation costs, and in cases where we only intend to evaluate a few new design cases, this cost will almost never be worth the large fixed cost. Thus, such methods may only apply to problems where effective Transfer Learning can take place or where data and training are effectively "free" as in Fig. 4 and 7.D. In this sense, our results mirror the somewhat pessimistic view taken by the conclusions in [34].

**Worth Experimenting With** However, if we relax the above conditions only slightly in one of several directions, then the picture changes significantly enough to warrant initial experiments in new problem domains. For example, if achieving up to an 80% RPI is acceptable for an initial design case, such as an approximate trade-space analysis or rough downselection of alternatives, then our results suggest that simple ID methods are efficient for this, breaking even with TO at anywhere from a small handful to a few hundred cases (on our admittedly simple example). Likewise, if your domain possesses existing datasets or specialized hardware that reduces the relative cost of data generation and training relative to optimization cost, then experimenting with possible gains from ID methods may be worthwhile.

**Almost Always** We saw several conditions under which training and using ID methods was almost always a good idea. Most

importantly, if we allow the ID method to work in concert with existing TO solvers as a warm start mechanism, there is comparatively little disadvantage to using ID methods, except in all but the most expensive of cases, and much to be gained. Figure 2's log-scaled x-axis shows that even simple models trained on a few data points can meaningfully accelerate TO convergence, even if their initial RPI is not great. In cases where we need to evaluate many possible new designs, Fig. 8 demonstrates clearly how ID methods reduce variable costs compared to only using TO, breaking even with them within a few hundred cases. While the specific break-even numbers we report are idiosyncratic to our example (and may be low compared to more complex problems), we suspect that similar behavior will occur in other problems, and that there will always exist a threshold wherein ID methods produce positive return on data investment.

Looking forward, we expect that ongoing and future work in improving both the sample efficiency and generalization ability ID methods will continue to shift these break even points lower over time. While there may be niche cases where investing in ID methods does not make sense—such as in large one-off optimization problems wherein generating data is neither practical nor warranted—we suspect that, given design's iterative nature, further ID advances will create a supportive interplay with existing optimizers that will be greater than the sum of their parts.

## ACKNOWLEDGEMENTS

We acknowledge the support from the National Science Foundation through award #1943699 as well as ARPA-E award DE-AR0001216. We also thank Jamie Guest, Katie Kirsch, and Amit Bhatia for helpful conversations on the chosen TO test problem and some of the cost measures.

## REFERENCES

[1] Topology optimisation of heat conduction problems governed by the poisson equation. http://www.dolfin-adjoint.org/en/latest/documentation/poisson-topology/poisson-topology.html. Accessed: 2022-02-10.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Ovo Adagha, Richard M Levy, and Sheelagh Carpendale. Towards a product design assessment of visual analytics in decision support applications: a systematic review. *Journal of Intelligent Manufacturing*, 28(7):1623–1633, 2017.

[4] Simon Arridge, Peter Maass, Ozan Öktem, and Carola-Bibiane Schönlieb. Solving inverse problems using data-driven models. *Acta Numerica*, 28:1–174, 2019.

[5] Martin P Bendsøe. Optimal shape design as a material distribution problem. *Structural optimization*, 1(4):193–202, 1989.

[6] Martin Philip Bendsøe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer methods in applied mechanics and engineering*, 71(2):197–224, 1988.

[7] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[8] Shai Bernard, Jun Wang, and Mark Fuge. Mean squared error may lead you astray when optimizing your inverse design methods. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 86229, page V03AT03A004. American Society of Mechanical Engineers, 2022.

[9] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] Nathan K Brown, Anthony P Garland, Georges M Fadel, and Gang Li. Deep reinforcement learning for engineering design through topology optimization of elementally discretized design domains. *Materials & Design*, 218:110672, 2022.

[11] Wojtek Buczynski, Fabio Cuzzolin, and Barbara Sahakian. A review of machine learning experiments in equity investment decision-making: Why most published research findings do not live up to their promise in real life. *International Journal of Data Science and Analytics*, 11(3):221–242, 2021.

[12] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[13] Aaditya Chandrasekhar and Krishnan Suresh. Tounn: topology optimization using neural networks. *Structural and Multidisciplinary Optimization*, 63(3):1135–1149, 2021.

[14] Qiuyi Chen, Jun Wang, Phillip Pope, Wei Chen, and Mark Fuge. Inverse design of two-dimensional airfoils using conditional generative models and surrogate log-likelihoods. *Journal of Mechanical Design*, 144(2):021712, 2022.

[15] Qiuyi Chen, Jun Wang, Phillip Pope, Mark Fuge, et al. Inverse design of two-dimensional airfoils using conditional generative models and surrogate log-likelihoods. *Journal of Mechanical Design*, 144(2), 2022.

[16] Wei Chen, Kevin Chiu, and Mark D Fuge. Airfoil design parameterization and optimization using bézier generative adversarial networks. *AIAA journal*, 58(11):4723–4735, 2020.

[17] Bo-Yu Chu, Chia-Hua Ho, Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. Warm start for parameter selection of linear classifiers. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 149–158, 2015.

[18] Gouri Deshpande, Guenther Ruhe, and Chad Saunders. How much data analytics is enough? the roi of machine learning classification and its application to requirements dependency classification. *arXiv preprint arXiv:2109.14097*, 2021.

[19] Ravi Hegde. Sample-efficient deep learning for accelerating photonic inverse design. *OSA Continuum*, 4(3):1019–1033, 2021.

[20] Martin Klaučo, Martin Kalúz, and Michal Kvasnica. Machine learning-based warm starting of active set methods in embedded model predictive control. *Engineering Applications of Artificial Intelligence*, 77:1–8, 2019.

[21] 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.

[22] Ruiwu Lei, Junqiang Bai, Hui Wang, Boxiao Zhou, and Meihong Zhang. Deep learning based multistage method for inverse design of supercritical airfoil. *Aerospace Science and Technology*, 119:107101, 2021.

[23] Mo-How Herman Shen Liang Chen. A New Topology Optimization Approach by Physics-Informed Deep Learning Process. *Advances in Science, Technology and Engineering Systems Journal*, 6(4):233–240, 2021.

[24] Qiyin Lin, Jun Hong, Zheng Liu, Baotong Li, and Jihong Wang. Investigation into the topology optimization for conductive heat transfer based on deep learning approach. *International Communications in Heat and Mass Transfer*, 97:103–109, 2018.

[25] Danny J Lohan, Ercan M Dede, and James T Allison. A study on practical objectives and constraints for heat conduction topology optimization. *Structural and Multidisciplinary Optimization*, 61(2):475–489, 2020.

[26] Keigo Nakamura and Yoshiro Suzuki. Deep learning-based topological optimization for representing a user-specified design area. *arXiv preprint arXiv:2004.05461*, 2020.

[27] Sangeun Oh, Yongsu Jung, Seongsin Kim, Ikjin Lee, and Namwoo Kang. Deep generative design: Integration of topology optimization and generative models. *Journal of Mechanical Design*, 141(11), 2019.

[28] John D Perkins, Tula R Paudel, Andriy Zakutayev, Paul F Ndione, Philip A Parilla, DL Young, Stephan Lany, David S Ginley, Alex Zunger, Nicola H Perry, et al. Inverse design approach to hole doping in ternary oxides: Enhancing p-type conductivity in cobalt oxide spinels. *Physical Review B*, 84(20):205207, 2011.

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

[30] Asmita Singh, Malka N Halgamuge, and Rajasekaran Lakshmiganthan. Impact of different data types on classifier performance of random forest, naive bayes, and k-nearest neighbors algorithms. *International Journal of Advanced Computer Science and Applications*, 8(12), 2017.

[31] Albert Tarantola. *Inverse problem theory and methods for model parameter estimation*. SIAM, 2005.

[32] Vicenc Torra. Trends in information fusion in data mining. *Information fusion in data mining*, pages 1–6, 2003.

[33] Dalei Wang, Cheng Xiang, Yue Pan, Airong Chen, Xiaoyi Zhou, and Yiquan Zhang. A deep convolutional neural network for topology optimization with perceptible generalization ability. *Engineering Optimization*, 54(6):973–988, 2022.

[34] Rebekka V Woldseth, Niels Aage, J Andreas Bærentzen, and Ole Sigmund. On the use of artificial neural networks in topology optimisation. *Structural and Multidisciplinary Optimization*, 65(10):1–36, 2022.

[35] Suna Yan, Fengwen Wang, and Ole Sigmund. On the non-optimality of tree structures for heat conduction. *International Journal of Heat and Mass Transfer*, 122:660–680, 2018.

[36] Junlian Yin and Dezhong Wang. Review on applications of 3d inverse design method for pump. *Chinese Journal of Mechanical Engineering*, 27(3):520–527, 2014.

[37] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pages 2528–2535. IEEE, 2010.

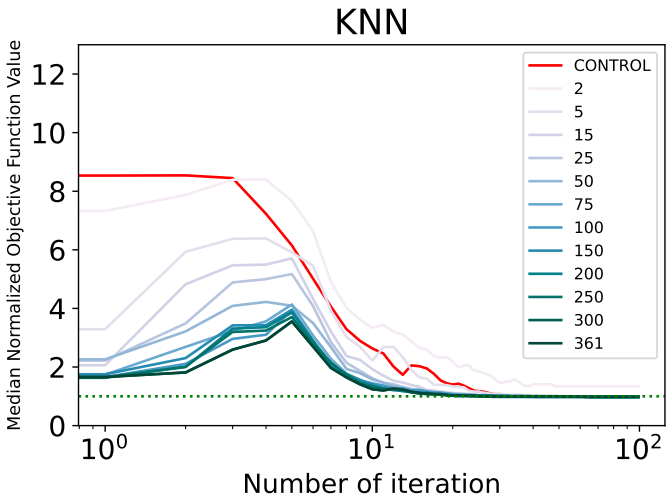


**FIGURE 9: THE EVOLUTION OF 2D HEAT CONDUCTION DESIGNS OVER THE COURSE OF THE OPTIMIZATION PROCESS. HERE, TRAJECTORY 2, 5, 15,....,300, AND 361 BELONGS TO THE TRAJECTORY INITIALIZED WITH THE PREDICTION OF THE KNN MODEL WITH THE CORRESPONDING SIZE OF TRAINING DATA.**

## SUPPLEMENTAL MATERIAL

Figure 9, 10 and 11 show the trajectories 2, 5, 15,....,300, and 361 belong to the trajectory initialized with the prediction of the KNN and RF model with the corresponding size of training data, respectively.

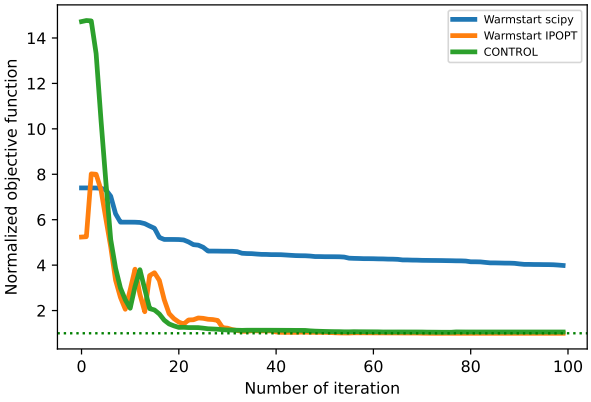


**FIGURE 12: COMPARISON OF DIFFERENT WARMSTART TO SOLVERS COMPARED TO THE CONTROL CONDITION.**

We used a KNN model prediction to warm start the optimizers for a design with a volume limit of 0.4 and an adiabatic length of 0.35. The comparison of warm start optimization with scipy and warm start optimization with IPOPT is shown in Fig 12.

,

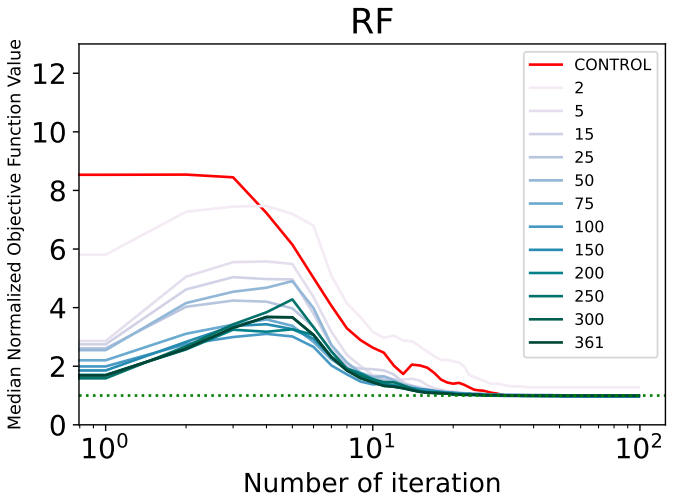


**FIGURE 10: THE EVOLUTION OF 2D HEAT CONDUCTION DESIGNS OVER THE COURSE OF THE OPTIMIZATION PROCESS. HERE, TRAJECTORY 2, 5, 15,....,300, AND 361 BELONGS TO THE TRAJECTORY INITIALIZED WITH THE PREDICTION OF THE RF MODEL WITH THE CORRESPONDING SIZE OF TRAINING DATA.**

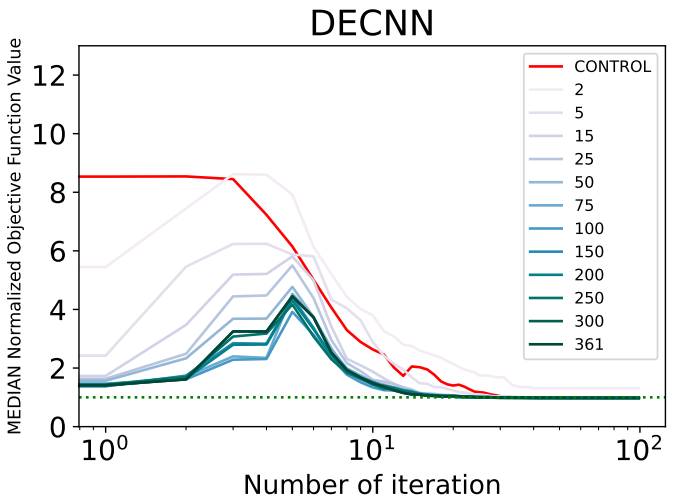


**FIGURE 11: THE EVOLUTION OF 2D HEAT CONDUCTION DESIGNS OVER THE COURSE OF THE OPTIMIZATION PROCESS. HERE, TRAJECTORY 2, 5, 15,....,300, AND 361 BELONGS TO THE TRAJECTORY INITIALIZED WITH THE PREDICTION OF THE DECNN MODEL WITH THE CORRESPONDING SIZE OF TRAINING DATA.**