Conservative Objective Models:
A Simple Approach to Effective Model-Based Optimization

Brandon Trabucco, Aviral Kumar, Xinyang Geng, Sergey Levine
Department of Computer Science
UC Berkeley
Berkeley, CA 94720, USA
{btrabucco,aviralk,young.geng,sergey.levine}@berkeley.edu

Abstract

In this paper, we aim to solve data-driven model-based optimization (MBO) problems, where the goal is to find an input that maximizes an unknown objective function provided access to only a static dataset of inputs and their corresponding score values. Such data-driven optimization procedures are the only applicable and scalable methods in many practical domains where active data collection is expensive (e.g., when optimizing over proteins) or dangerous (e.g., when optimizing over aircraft designs). Typical methods for MBO that optimize the input against a learned model of the unknown score function are affected by erroneous overestimation in the learned model that drives the optimizer to low-scoring or even invalid inputs. To overcome this problem, we propose *conservative objective models* (COMs), a method that obtains estimators that are robust to out-of-distribution inputs. In practice, COMs outperform existing methods on a wide range of MBO problems over neural network controller weights, robot morphologies, molecule substructures, superconducting materials, and proteins, in high-dimensional cases achieving a performance of **1.3-2x** times the best existing method.

1 Introduction

Black-box model-based optimization (MBO) problems are ubiquitous in a wide range of domains such as protein [1] or molecule design [2], designing controllers [3] or robot morphologies [4], optimizing neural network designs [5] and aircraft design [6]. Existing methods to solve such model-based optimization problems often rely on a tight interaction loop between optimization and *active* data collection [7]. Active data collection can be expensive or even dangerous: evaluating a real design involves complex real-world procedures that must be performed under special conditions. While these problems can be solved by building simulators, mimicking reality is often impossible. Therefore, making progress on a broad range of MBO problems requires developing *data-driven* methods that can obtain optimized designs by training highly generalizable and expressive deep neural network models on a previously collected dataset of inputs ($\mathbf{x}$) and their corresponding objective values (y), without access to the true function or any form of active data collection [8]. Moreover, in a number of these practical domains, such as protein [9] or molecule design [2], plenty of prior data already exists and it can be utilized for completely offline model-based optimization.

Typical approaches for addressing MBO problems learn some sort of a *model* of the unknown objective function ($\hat{f}(\mathbf{x})$) that maps an input [7] (or a representation of the input [10]) to its objective value via supervised regression, and then optimize the input against this learned model via, for instance, gradient ascent. For MBO problems where the space of valid inputs forms a narrow manifold in a high-dimensional space, any overestimation errors in the learned model will drive the optimization procedure towards off-manifold, invalid and low-scoring inputs [8], as these will falsely

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2020., Vancouver, Canada.

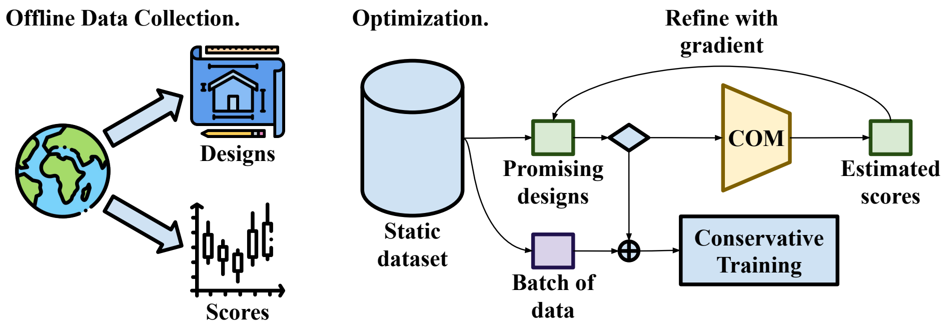

Figure 1: **Diagram of training and optimization using COMs.** The section on the left indicates that each task provides a static dataset that is collected offline without an MBO algorithm in-the-loop. The section on the right shows how an objective model is used to refine promising designs using gradient ascent, and how these designs are inputs to a conservative regularizer.

appear optimistic under the learned model. Since the procedure has no active data collection, the algorithm is unable to recover from poor solutions which it erroneously predicts are optimal.

If we can instead learn a *conservative* model of the objective function, which represents a robust version of the true function even on out-of-distribution inputs, optimizing against this conservative model would produce solutions that perform well with respect to the true function, while avoiding the aforementioned overestimation issue. In this paper, we propose a method to learn such *conservative objective models* (COMs), and then optimize the design against this conservative function using a very simple gradient-ascent procedure. Building on recent advances in offline reinforcement learning [11, 12], the key idea behind our method is to train the learned score model with an additional objective that maximizes the expected function value on inputs observed in the dataset and minimizes the expected function value on out-of-distribution designs, generated using a specially designed adversarial procedure. Further we use specially-designed variants of typically used ensemble techniques to then perform hyperparameter selection for COMs, effectively making the void of any new hyperparmeter.

The primary contribution of this paper is a novel approach for addressing data-driven model-based optimization problems by learning a conservative model of the unknown objective function that is robust to out-of-distribution inputs, and then optimizing the input against this conservative model via an extremely simple gradient ascent procedure. COMs are simple to implement: they only require training a model for the objective function, as opposed to other recent approaches that also train generative models. By using specially designed techniques for offline hyperparameter selection, we empirically demonstrate the efficacy of COMs on six complex MBO tasks that span a wide range of real-world tasks including protein and molecule substructure design, neural network parameter optimization, robot morphology design, and superconducting material design. On some tasks, COMs outperform the *best* existing method *on that task* by a factor of **1.3-2x**.

2 Preliminaries

The goal in data-driven, offline model-based optimization [8] is to find best possible or near-optimal solution, $\mathbf{x}^*$, to optimization problems of the form

$$\mathbf{x}^* \leftarrow \arg\max_{\mathbf{x}} \; f(\mathbf{x}), \tag{1}$$

where $f(\mathbf{x})$ is an unknown (possibly stochastic) objective function. The MBO algorithm is provided access to a static dataset $\mathcal{D}$ of inputs and their objective values, $\mathcal{D} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$. While a variety of MBO methods have been developed [10, 1, 8, 13], most methods for tackling MBO problems fit a parametric model to the samples of the true objective function in $\mathcal{D}$, $\hat{f}_\theta(\mathbf{x})$, via supervised training: $\hat{f}_\theta(\mathbf{x}) \leftarrow \arg\min_\theta \sum_i (\hat{f}_\theta(\mathbf{x}_i) - y_i)^2$, and find $\mathbf{x}^*$ in Equation 1 by optimizing $\mathbf{x}$ against this learned model $\hat{f}_\theta(\mathbf{x})$, typically with some mechanism to additionally minimize distribution shift. Since closed-form optimization solutions may not exist when complex parametric models such as deep neural networks are used, a common choice for optimizing $\mathbf{x}$ in Equation 1 is gradient

Algorithm 1 COM: Training Conservative Models

```
Initialize $\hat{f}_\theta$. Partition $\mathcal{D}$ into validation data, $\mathcal{D}_{\text{val}}$, and train data $\mathcal{D} \setminus \mathcal{D}_{\text{val}}$. Pick $T(T_{\text{train}}), \eta, \alpha$.
for $i = 1$ to training_steps do
    Sample $(\mathbf{x}_0, y) \sim \mathcal{D}$
    Find $\mathbf{x}_T$ via gradient ascent from $\mathbf{x}_0$:
    $\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla_\mathbf{x} \hat{f}_\theta(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_t}$.
    Minimize $\mathcal{L}(\theta; \alpha)$ with respect to $\theta$.
    $\mathcal{L}(\theta; \alpha) = (\hat{f}_\theta(\mathbf{x}) - y)^2 - \alpha \hat{f}_\theta(\mathbf{x}) + \alpha \hat{f}_\theta(\mathbf{x}_T)$
    $\theta \leftarrow \theta - \lambda_\theta \nabla_\theta \mathcal{L}(\theta; \alpha)$
end for
```

Algorithm 2 COM: Finding $\mathbf{x}^\star$

```
1: Select $N$ values of $\alpha$: $\{\alpha_1, \cdots, \alpha_N\}$.
2: Run Algorithm 1 for each $\alpha_i$, to obtain the corresponding $\hat{f}^\star_\theta(i)$.
3: Select $\alpha^*$ using Equation 6 and let the corresponding function be denoted as $\hat{f}^\star_\theta$.
4: Initialize optimizer at the optimum in $\mathcal{D}$:
5:     $\tilde{\mathbf{x}} = \arg\max_{(\mathbf{x},y)\in\mathcal{D}} \; y$
6: Find $\mathbf{x}^\star$ via gradient ascent from $\tilde{\mathbf{x}}$:
7:     $\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla_\mathbf{x} \hat{f}^\star_\theta(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_t}$.
8: Return the solution $\mathbf{x}^\star = \mathbf{x}_{T_{\text{eval}}}$.
```

descent on the learned function as given by

$$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha \nabla_x \hat{f}_\theta(\mathbf{x})|_{x=\mathbf{x}_k}, \quad \text{for } \; k \in [1, T], \quad \mathbf{x}^\star = \mathbf{x}_T. \tag{2}$$

The fixed point of the above procedure $\mathbf{x}_T$ is then the output of the MBO procedure. In high-dimensional input spaces, where valid $\mathbf{x}$ values lie on a narrow manifold in a high-dimensional space, such an optimization procedure is prone to producing low-scoring inputs, which may not even be valid. This is because $\hat{f}$ may erroneously overestimate objective values at out-of-distribution points which would naturally lead the optimization to such invalid points. For notational convenience let, $\hat{D}(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{D}} \delta_{\mathbf{x}=x_i}$ denote the empirical distribution of inputs $\mathbf{x}$ in the dataset.

3 Conservative Objective Models for Model-Based Optimization

In this section, we present our approach, conservative objective models (COMs). COMs learn estimates of the true function that is robust to out-of distribution inputs. Robustness of the objective function prevents erroneous overestimation that would drive the optimizer (Equation 2) to produce out-of-distribution and low-scoring inputs, giving rise to stable and robust optimization behavior. We first discuss a procedure for learning such robust estimates and explain how these conservative models can be used for MBO, and then discuss a simple, ensembling-based procedure for hyperparameter selection that makes COMs fully amenable to completely offline training.

Learning conservative objective models (COMs). Building on recent methods from offline RL [12], the key idea behind our approach is to augment the training of a learned objective model, $\hat{f}_\theta(\mathbf{x})$, with a regularizer that minimizes the expected value of this function under a specially chosen adversarial distribution $\mu(\mathbf{x})$, while also maximizing the expected value of this function under the empirical distribution of the inputs $\mathbf{x}$, $\hat{D}(\mathbf{x})$, in the dataset $\mathcal{D}$. We will discuss the choice of $\mu(\mathbf{x})$ in the next part of this section. Formally, this objective is given by the following equation, where α is a parameter that trades off conservatism for regression.

$$\hat{f}^\star_\theta \leftarrow \arg\min_{\theta \in \Theta} \; \alpha \left(\mathbb{E}_{\mathbf{x} \sim \mu(\mathbf{x})} \left[\hat{f}_\theta(\mathbf{x}) \right] - \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\hat{f}_\theta(\mathbf{x}) \right] \right) + \frac{1}{2} \mathbb{E}_{(\mathbf{x},y) \sim \mathcal{D}} \left[\left(\hat{f}_\theta(\mathbf{x}) - y \right)^2 \right], \tag{3}$$

The value of α and the choice of distribution $\mu(\mathbf{x})$ play a crucial role in determining the utility of this bound. For instance, if $\mu(\mathbf{x})$ is not chosen carefully, then the learned function $\hat{f}^\star_\theta$ may not be a lower-bound on the $\mathbf{x}$ of interest (for example, the value of $\mathbf{x}$ found by the optimization procedure). Similarly, if the chosen α is very small, then the resulting $\hat{f}^\star_\theta(\mathbf{x})$ may not be a robust estimate of the actual function $f(\mathbf{x})$, whereas if the chosen α is too large, then the learned function will behave primarily as a discriminator between $\mu(\mathbf{x})$ and $\hat{D}(\mathbf{x})$, and be less useful for optimization. We will how to choose $\mu(\mathbf{x})$ and α in this section, but we first discuss how $\hat{f}$ can be used for optimization.

Using COMs for offline model-based optimization. Once we have obtained a conservative model of the score function from Equation 3, we must use this learned model for finding the best possible input, $\mathbf{x}^\star$. While prior works [8, 1] use $\hat{f}^\star_\theta$ in conjunction with generative models, we find it sufficient to run T iterations of naïve gradient ascent on $\hat{f}^\star_\theta$, starting from the best point $\mathbf{x}_0 \in \mathcal{D}$, as shown in Equation 4 (and in Algorithm 2, Line 6):

$$\mathbf{x}^\star = \mathbf{x}_T, \quad \text{where} \quad \mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla_\mathbf{x} \hat{f}^\star_\theta(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_t} \; \forall \, t \in [1, \cdots, T] \text{ and } \mathbf{x}_0 \sim \mathcal{D}. \tag{4}$$

Choosing $\mu(\mathbf{x})$. Since our goal is to use the conservative objective model $\hat{f}^\star_\theta$ for optimization, as dictated by Equation 4, we are primarily interested in preventing erroneous overestimation on $\mathbf{x}_T$ generated from the learned objective model. As a result, our choice of $\mu(\mathbf{x})$ specifically selects candidate points $\mathbf{x}$ that can be generated by performing T steps of gradient ascent starting from any data point $\mathbf{x}_0$ sampled from the dataset, on the current instance of $\hat{f}_\theta$. To convert this into a valid training objective, we design a minimax optimization problem that selects $\mu(\mathbf{x})$ in the manner described above, which is given below, with changes from Equation 3 indicated in red:

$$\min_{\theta \in \Theta} \max_{\mu(\mathbf{x}_T|\mathbf{x}_0)} \mathcal{L}(\theta, \mu; \alpha) := \alpha \left(\mathbb{E}_{\mathbf{x}_0 \sim \mathcal{D}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)} \left[\hat{f}_\theta(\mathbf{x}_T) \right] - \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\hat{f}_\theta(\mathbf{x}) \right] \right)$$
$$+ \frac{1}{2} \mathbb{E}_{(\mathbf{x},y) \sim \mathcal{D}} \left[\left(\hat{f}_\theta(\mathbf{x}) - y \right)^2 \right] + \frac{1}{2\eta} \mathbb{E}_{\mathbf{x}_0 \sim \mathcal{D}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)} \left[||\mathbf{x}_0 - \mathbf{x}_T||_2^2 \right]. \quad (5)$$

Choosing α. The remaining piece in converting the procedure discussed in Equations 3 and 5 into a viable algorithm is to devise a scheme that automatically selects the values of the hyperparameter α. As discussed previously, if α is too large, $\hat{f}^\star_\theta$ is expected to behave like a discriminator, and not as an objective function, since it would assign higher values to inputs in the dataset, and low values to *all* other inputs even if they have high objective values. Automating the selection of α is hard, since its value depends strongly on the magnitude of the groundtruth objective, which is unknown in general. Instead, we use a modified training procedure in Equation 3 or 5 that poses Equation 3 as a constrained optimization problem with α assuming the role of a Lagrange dual variable towards satisfying a constraint that controls the difference in values of the learned objective under $\mu(\mathbf{x})$ and $\mathcal{D}(\mathbf{x})$. This is formally captured as solving the following optimization problem:

$$\hat{f}^\star_\theta \leftarrow \arg\min_{\theta \in \Theta} \frac{1}{2} \mathbb{E}_{(\mathbf{x},y) \sim \mathcal{D}} \left[\left(\hat{f}_\theta(\mathbf{x}) - y \right)^2 \right] \text{ s.t. } \left(\mathbb{E}_{\mathbf{x} \sim \mu(\mathbf{x})} \left[\hat{f}_\theta(\mathbf{x}) \right] - \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\hat{f}_\theta(\mathbf{x}) \right] \right) \leq \tau. \quad (6)$$

While Equation 6 introduces a new hyperparameter τ in place of α, we argue that τ can uniformly be controlled over all he tasks without considerations for the landscape of the groundtruth objective function. Empirically, we demonstrate in Section 5 that COMs are considerably robust to the values of $\tau \leq 0.1$, and a single value of $\tau = 0.05$ is effective on *every* task.

Selecting optimized designs $\mathbf{x}^\star$. So far we have discussed how COMs can be trained and used for optimization, however we have not established a way to determine which $\mathbf{x}_t$ (Equation 4) encountered in the optimization trajectory should be used as our final solution $\mathbf{x}^\star$. Simply picking a random $\mathbf{x}_t \sim \mathcal{X}_{\text{opt}} := \{\mathbf{x}_1, \cdots, \mathbf{x}_{T_{\text{eval}}}\}$ is not sufficient as performance is generally highly variable within this set $\mathcal{X}_{\text{opt}}$ (see Section 5), which motivates the need for an offline model-selection strategy to select promising $\mathbf{x}_t$ samples. Our mechanism for model-selection uses an ensemble of separately trained objective models to rank the performance of different $\mathbf{x}_t$ values obtained during gradient ascent and we then return $\mathbf{x}^\star$ that performs the best in terms of its prediction under this ensemble of models. Concretely, our ensemble consists of N independently trained standard objective models, $(\tilde{f}_1(\mathbf{x}), \cdots, \tilde{f}_N(\mathbf{x}))$ and we use the minimum predicted value over the ensemble, $\tilde{f}(\mathbf{x}) := \min(\tilde{f}_1(\mathbf{x}), \cdots, \tilde{f}_N(\mathbf{x}))$ as the ranking statistic for input iterates $\mathbf{x}_1, \cdots, \mathbf{x}_{T_{\text{eval}}}$ obtained via gradient ascent on the COM for large enough values of T_{eval}. We then return $\mathbf{x}^\star := \arg\max_{\mathbf{x}} \tilde{f}(\mathbf{x})$ that maximizes the ranking statistic as the solution to the optimization problem.

Overall algorithm and practical implementation. Finally, we combine the individual components discussed above to obtain a complete algorithm for offline model-based optimization. Pseudocode for our algorithm is shown in Algorithm 1. COMs parameterize the objective model, $\hat{f}_\theta(\mathbf{x})$, via a feed-forward neural network with parameters θ. Our method then alternates between approximately generating samples $\mathbf{x}_{T_{\text{train}}} \sim \mu(\mathbf{x})$ via gradient ascent (Line 4), and optimizing parameters θ using Equation 5 (Lines 7 and 8). Crucially, note that when optimizing θ using gradient ascent (Line 6), we utilize the complete derivative w.r.t. θ and do not stop gradient flow through the θ-dependent process that generates $\mathbf{x}_T$ starting from $\mathbf{x}_0$. Finally, at the end of training, we run the gradient ascent procedure over the learned objective model $\hat{f}^\star_\theta(\mathbf{x})$ for a large T_{eval} number of gradient steps and then use our specially designed model-selection strategy to select one of the inputs encountered in the optimization trajectory $\mathcal{X}_{\text{opt}} = (\mathbf{x}_1, \cdots, \mathbf{x}_{T_{\text{eval}}})$ as our final solution $\mathbf{x}^\star$.

Implementation details. For all of our experiments, the conservative objective model $\hat{f}_\theta$ is modeled as a neural network with two hidden layers of size 2048 each and leaky ReLU activations. More

	$\mathcal{D}$ (best)	**MINs**	**CbAS**	**Autofocus**	**Grad. ascent**	**COMs (ours)**
A	3.152	3.315 ± 0.033	3.408 ± 0.029	3.365 ± 0.023	2.894 ± 0.001	*3.305 ± 0.024*
B	6.558	6.508 ± 0.236	6.301 ± 0.131	6.345 ± 0.141	6.401 ± 0.186	**6.876 ± 0.128**
C	73.90	80.23 ± 10.67	72.17 ± 8.652	77.07 ± 11.11	89.64 ± 9.201	**110.0 ± 6.804**
D	1361.6	746.1 ± 636.8	547.1 ± 423.9	443.8 ± 142.9	1050.8 ± 284.5	**2395.7 ± 561.7**
E	108.5	388.5 ± 9.085	393.0 ± 3.750	386.9 ± 10.58	399.9 ± 4.941	378.8 ± 10.01
F	215.9	352.9 ± 38.65	369.1 ± 60.65	376.3 ± 47.47	390.7 ± 49.24	341.4 ± 28.47

Table 1: **Comparative evaluation of COMs** against prior methods in terms of the mean 100%-percentile score and its standard deviation over 16 trials. Task letters correspond to (A) GFP-v0, (B) MoleculerActivity-v0, (C) Superconductor-v0, (D) HopperController-v0, (E) AntMorphology-v0, and (F) DkittyMorphology-v0, standard tasks provided by [15]. COMs perform strictly better on high-dimensional tasks, obtaining about **2x** gains on HopperController-v0, and compelling gains on MoleculeActivity-v0 and Superconductor-v0 tasks. In addition, COMs is shown to be stable across domains, and is the only method that is able to consistently find solutions that outperform the best training point for each task, given by $\mathcal{D}$ **(best)**.

details on the network structure can be found in Appendix A. In order to train this conservative objective model, we use the Adam optimizer [14] with a learning rate of 10^{-3}. Empirically, we found larger values of η to produce $\mathbf{x}_T$ with extremely low values of $\hat{f}^\star_\theta(\mathbf{x}_T)$ and thus selected the largest η that avoided this phenomenon in practice. We utilize the ensemble-based model-selection scheme to select the optimized input $\mathbf{x}^\star$ obtained out of set $\mathcal{X}_{\text{opt}}$ obtained by running gradient ascent for a large $T_{\text{eval}} = 500$. Crucially, our ensembles consist of 8 different neural networks with varying activation functions (more details in Appendix B), which we found to be key in enabling effective model-selection for $\mathbf{x}^\star$. A naïve ensembling scheme which consists of neural network $\tilde{f}_i(\mathbf{x})$ with the same architecture and activation function only produced generally selected $\mathbf{x}^\star = \mathbf{x}_{T_{\text{eval}}}$ which generally had worse performance. We set $\tau = 0.05$ uniformly across all tasks, and use Lagrangian dual descent for automatic tuning of the Lagrange multiplier α.

4 Related Work

We now briefly discuss prior works in model-based optimization, including prior work on active model-based optimization and work that utilizes offline datasets for data-driven MBO.

Bayesian optimization. Most prior work on model-based optimization has focused on the active setting, where derivative free methods such as the cross-entropy method [16] and other methods derived from the REINFORCE trick [17, 18], reward-weighted regression [19], and Gaussian processes [20, 21, 7] have been utilized. Most of these methods focus mainly on low-dimensional tasks with active data collection. Practical approaches have combined these methods with Bayesian neural networks [20, 7], latent variable models [22, 23, 24], and ensembles of learned score models [25, 26, 27]. These methods still require actively querying the true function $f(\mathbf{x})$. Further, as shown by [1, 13, 8], these Bayesian optimization methods are susceptible to producing invalid out-of-distribution inputs in the offline setting. Unlike these methods, COMs are specifically designed for the offline setting with high-dimensional inputs, and avoid out-of-distribution inputs.

Offline model-based optimization. Recent works have also focused on optimization in the completely offline setting. Typically these methods utilize a deep generative model [28, 29] that models the manifold of inputs. [1, 13] use a variational autoencoder [28] to model the space of $\mathbf{x}$ and use it alongside a learned objective function. [8] learn an inverse map from the scalar objective y to input $\mathbf{x}$ and search for the optimal one-dimensional y during optimization. Modeling the manifold of valid inputs globally can be extremely challenging (see Ant, Hopper and DKitty results in Section 5), and as a result these generative models often need to be tuned for each domain [15]. In contrast, COMs do not require any generative model, and instead fit an approximate objective function with a simple regularizer, providing both a simpler algorithm and better empirical performance.

Adversarial examples. As discussed in Section 2, MBO methods based on learned objective models naturally query the learned function on "adversarial" inputs, where the learned function erroneously overestimates the true function. This is superficially similar to adversarial examples in supervised learning [30] which can be generated by maximizing the input against the loss function. While

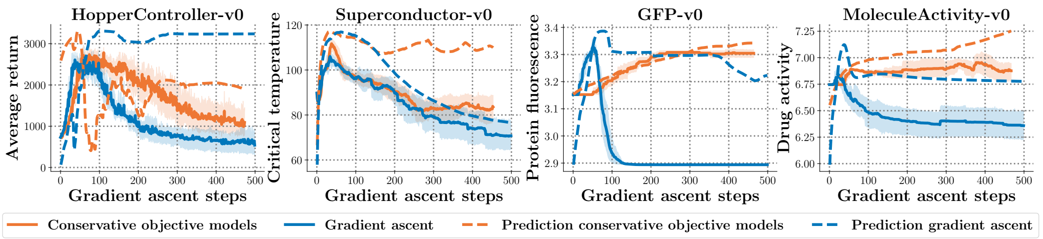

Figure 2: **Stability of conservative score models versus naive gradient ascent.** The x-axis shows the number of gradient ascent steps taken on the design X^* and the y-axis shows the 100th percentile of the ground truth task score function evaluated at every gradient step. The orange line corresponds to the performance of samples from COMs, whereas the dark blue line corresponds to a naive gradient ascent optimizer. Predictions of an ensemble of models that are not used for gradient calculation are shown with dashed lines of the same color. In every case, COMs reaches solutions that remain at higher performance for longer—indicating COMs have improved stability.

adversarial examples have been formalized as out-of-distribution inputs lying in the vicinity of the data distribution and prior works have attempted to correct for them by encouraging smoothness [31] of the learned function, and there is evidence that robust objective models help mitigate over estimation [32], these solutions may be ineffective in MBO settings when the true function is itself non-smooth. Instead making conservative predictions on such adversarially generated inputs may prevent poor performance.

5 Experimental Evaluation

The goal of our experimental evaluation is to evaluate the efficacy of COMs for offline model-based optimization. To this end, we first perform a comparative evaluation of COMs on six offline MBO tasks based on problems in physical sciences, neural network design, and biology proposed in the design-bench benchmark [15]. Then, we perform an empirical analysis on COMs that aims to answer the following questions: **(1)** How effective is the ensembling strategy discussed in Section 3 for hyperparameter selection?, **(2)** Is the conservative training loss essential for preventing optimization from being driven to out-of-distribution inputs?, and **(3)** Are COMs robust to hyperparameter choices? We answer these questions by comparing standard COMs and COMs when various components of the method are turned off.

5.1 Empirical Performance on Benchmark Tasks

We first compare COMs to a range of recently proposed methods for offline MBO in high-dimensional input spaces: CbAS [1], MINs [8] and autofocused CbAS [13], that augments CbAS with a reweighted objective model. CbAS and MINs train generative models such as VAEs [28] and GANs [29], which generally require task-specific neural net architectures as compared to substantially simpler COMs. Moreover, generative models can be hard to train in many cases, for example, a VAE/GAN can be unstable when optimizing over discrete input spaces.

Our evaluation protocol is drawn follows prior work [1, 15]: we query each method to obtain the top $N = 128$ most promising optimized samples according to the model $\mathbf{x}_1^\star, \cdots, \mathbf{x}_N^\star$, and then report the 100^{th} percentile ground truth objective values on this set of samples, $\max(\mathbf{x}_1^\star, \cdots, \mathbf{x}_N^\star)$ averaged over 16 trials. We would argue that such an evaluation scheme is reasonable as it is typically followed in real-world MBO problems where a set of optimized inputs are produced by the model, and the best performing one of them is finally used for evaluation.

Our results for different domains are shown in Table 1. The set of domains studied includes a combination of those studied in prior work: **(i)** GFP-v0 [1], where the goal is to optimize over 238-dimensional protein sequences to maximize fluorescence using a dataset of 5000 points, **(ii)** HopperController-v0 [8], where the goal is to optimize over 5126-dimensional weights of a neural network policy on the Hopper-v2 gym domain using a dataset of 3200 points, **(iii)** Superconductor-v0 [13], where the goal is to optimize over 81-dimensional superconductor designs to maximize the critical temperature using 16953 points, **(iv)** Ant and DKittyMorphology-v0, where the goal is to

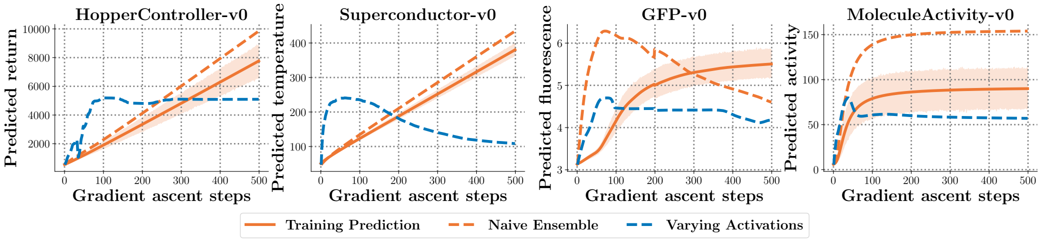

Figure 3: **Predictions of different models out-of-distribution.** In each plot, the predictions of a model whose input is optimized with gradient ascent is shown in solid orange, and the predictions of two ensemble methods are shown with dashed lines. The Naive Ensemble trains with different bootstraps, but uses the same architecture as the training model. The strategy labelled Varying Architecture corresponds to our ensembling strategy. The x-axis indicates the number of gradient ascent steps taken on the design $x*$. The y-axis corresponds to model predicted scores.

design the 56-dimensional morphology of robots to maximize policy performance using datasets of size 12300 and 9546 respectively, and **(v)** MoleculeActivity-v0 [33], where the goal is to design 1024-dimensional substructure fingerprints to maximize activity with a target assay using a dataset of size 4216. Results for baseline methods are based on numbers reported by [15]. Additional details for the setup of each offline MBO task is provided in Appendix Section C.

Observe in Table 1, that on all tasks COMs either attain the best results or attain results comparable with the best performing method on that task. In addition, COMs are shown to be the only method to attain higher performance that the best training point on every task. While COMs perform similarly to CbAS and MINs on GFP-v0 and DKittyMorphology-v0, it substantially outperforms prior methods on the high-dimensional HopperController-v0 and Superconductor-v0 tasks, by a factor of **1.3-2x**. A naïve objective model without the conservative term, which is prone to falling off-the-manifold of valid inputs, indeed performs poorly in all cases. While combining this naïve model with our proposed model-selection strategy improves its performance greatly, it is still worse as compared to COMs in each case. These results indicate that COMs can serve as simple yet powerful method for offline MBO across a variety of domains. Furthermore, note that COMs only require training a parametric model $y = \hat{f}(x)$ of the objective function with a regularizer, without any need for training a generative model, which may be harder in practice to tune.

5.2 Ablation Studies

In this section, we perform an experimental analysis of COMs to answer questions posed at the beginning of Section 5. First we evaluate the efficacy of the proposed ensembling scheme for selecting the optimized solution $\mathbf{x}^\star$. In order to better compare the optimization trajectories from different methods, we visualize the true objective value for each $\mathbf{x}_t$ encountered during gradient-ascent in Figure 2. Observe that a naïve objective model can attain good performance for a specific "hand-tuned" number of ascent steps, and applying our model-selection scheme on this nav̈e model gives a boost in its performance in some cases. Furthermore, note that a naïve ensemble that does not vary activation functions in the ensemble has its predictions continually increase in Figure 3 and typically selects the final $\mathbf{x}_t$, leading to worse performance. This evidence strongly indicates that effective model-selection is crucial to the success of MBO methods that learn an objective model and optimize with respect to it's inputs, to which our proposed scheme is effective.

Next, we evaluate the necessity of conservatism induced by COMs by comparing COMs in Figure 2 to a naïve objective model equipped with our proposed model-selection scheme. Observe that COMs are considerably more robust than a naïve objective models, and significantly many $\mathbf{x}_t$ values in their optimization trajectory attain good performance. Why is this important? Observe in Figure 2, that while our model-selection strategy is reasonably effective in removing a hand-tuned number of ascent steps, it does not precisely select $\mathbf{x}_t$ that are optimal, but selects points that lie *close* to peak performance. If overestimation errors in a naïve learned model drive the optimizer to out-of-distribution $\mathbf{x}_t$ values within a few ascent steps, it seems unlikely that we will be able to perform robust and stable optimization with naïve models even with sophisticated model-selection schemes. On the other hand, our conservative method is relatively stable at peak performance giving rise to more robust optimization performance than prior methods.

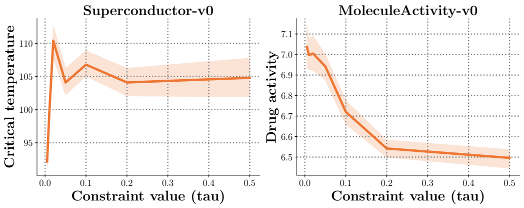

Figure 4: **Ablation of sensitivity to the conservative threshold τ.** Performance is stable for a small range of τ. This figure shows performance (y-axis) evaluated using our offline stopping criterion that leverages an ensemble's predictions, versus several values of the conservatism threshold τ (x-axis). The value chosen in our experiments is $\tau = 0.05$ which works well in both of these examples. While performance decays when τ is too large, for values of τ close to 0.05 remain competitive with the results we report in Table 1.

Finally, we evaluate the sensitivity of COMs to the value of τ chosen universally across environments by comparing the sensitivity of performance of COMs with respect to τ on two tasks (Superconductor and MoleculeActivity) in Figure 4. Note that in both cases, a single value of $\tau = 0.05$ gives rise to effective optimization performance. Hence we can conclude that while COMs do not eliminate the hyperparameter τ, τ is a *domain-independent* hyperparameter which can be fixed to a constant value, such as $\tau = 0.05$, a value strongly supported by the plots in Figure 4.

6 Discussion

We proposed conservative objective models (COM), a simple method for offline model-based optimization, that learns a conservative estimate of the actual objective function and optimizes the input against this estimate. Empirically, COMs are more stable than prior MBO methods, returning solutions that are comparable to and even better than the best existing MBO algorithms on six benchmark tasks. In this evaluation, COMs are consistently high performing, and in the most high-dimensional cases, COMs improves on the next best method by a factor of **1.3-2x**. The simplicity of COMs combined with their empirical strength make them a promising optimization backbone to find solutions to challenging and high-dimensional offline MBO problems. In contrast to certain prior methods, COMs are designed to mitigate overestimation of out-of-distribution inputs, and show an improvement to stability once good solutions to the optimization problem are found. Coupled with our effective model-selection criterion, the proposed algorithm is fully offline.

While our results suggest that COMs are effective on a number of MBO problems, there exists room for improvement. The somewhat naïve gradient-ascent optimization procedure employed by COMs can likely be improved by combining it with manifold modelling techniques, which can accelerate optimization by alleviating the need to traverse the raw input space. Similar to offline RL and supervised learning, learned objective models in MBO are prone to overfitting, especially in limited data settings. Understanding different mechanisms by which overfitting can happen and correcting for it is likely to greatly amplify the applicability of COMs to a large set of practical MBO problems that only come with small datasets. Finally, despite improvements to stability provided by COMs, a model-selection scheme is needed because samples eventually fall off-manifold. Understanding why and how samples found by gradient ascent become off-manifold could result in a more powerful gradient-ascent optimization procedure that does not require a model-selection scheme.

References

[1] D. Brookes, H. Park, and J. Listgarten, "Conditioning by adaptive sampling for robust design," in *Proceedings of the 36th International Conference on Machine Learning*, PMLR, 2019.

[2] A. Gaulton, L. J. Bellis, A. P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani, *et al.*, "Chembl: a large-scale bioactivity database for drug discovery," *Nucleic acids research*, vol. 40, no. D1, pp. D1100–D1107, 2012.

[3] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with gaussian processes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 491–496, IEEE, 2016.

[4] T. Liao, G. Wang, B. Yang, R. Lee, K. Pister, S. Levine, and R. Calandra, "Data-efficient learning of morphology and controller for a microrobot," in *2019 IEEE International Conference on Robotics and Automation*, 2019.

[5] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017.

[6] W. Hoburg and P. Abbeel, "Geometric programming for aircraft design optimization," vol. 52, 04 2012.

[7] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, 2012.

[8] A. Kumar and S. Levine, "Model inversion networks for model-based optimization," *arXiv preprint arXiv:1912.13464*, 2019.

[9] K. S. Sarkisyan, D. A. Bolotin, M. V. Meer, D. R. Usmanova, A. S. Mishin, G. V. Sharonov, D. N. Ivankov, N. G. Bozhanova, M. S. Baranov, O. Soylemez, N. S. Bogatyreva, P. K. Vlasov, E. S. Egorov, M. D. Logacheva, A. S. Kondrashov, D. M. Chudakov, E. V. Putintseva, I. Z. Mamedov, D. S. Tawfik, K. A. Lukyanov, and F. A. Kondrashov, "Local fitness landscape of the green fluorescent protein," *Nature*, vol. 533, pp. 397–401, May 2016.

[10] R. Gómez-Bombarelli, D. Duvenaud, J. M. Hernández-Lobato, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," in *ACS central science*, 2018.

[11] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[12] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *arXiv preprint arXiv:2006.04779*, 2020.

[13] C. Fannjiang and J. Listgarten, "Autofocused oracles for model-based design," *arXiv preprint arXiv:2006.08052*, 2020.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[15] Anonymous, "Design-bench: Benchmarks for data-driven offline model-based optimization," in *Submitted to International Conference on Learning Representations*, 2021. under review.

[16] R. Y. Rubinstein and D. P. Kroese, *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-carlo Simulation (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2004.

[17] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[18] R. Y. Rubinstein, "Optimization of computer simulation models with rare events," *European Journal of Operations Research*, vol. 99, pp. 89–112, 1996.

[19] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, 2007.

[20] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable bayesian optimization using deep neural networks," in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, 2015.

[21] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, “Taking the human out of the loop: A review of bayesian optimization,” *Proceedings of the IEEE*, vol. 104, pp. 148–175, 2016.

[22] H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh, “Attentive neural processes,” in *International Conference on Learning Representations*, 2019.

[23] M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. M. A. Eslami, and Y. W. Teh, “Neural processes,” *CoRR*, vol. abs/1807.01622, 2018.

[24] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. M. A. Eslami, “Conditional neural processes,” in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, 2018.

[25] C. Angermueller, D. Belanger, A. Gane, Z. Mariet, D. Dohan, K. Murphy, L. Colwell, and D. Sculley, “Population-based black-box optimization for biological sequence design,” *arXiv preprint arXiv:2006.03227*, 2020.

[26] C. Angermueller, D. Dohan, D. Belanger, R. Deshpande, K. Murphy, and L. Colwell, “Model-based reinforcement learning for biological sequence design,” in *International Conference on Learning Representations*, 2020.

[27] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae, *et al.*, “Chip placement with deep reinforcement learning,” *arXiv preprint arXiv:2004.10746*, 2020.

[28] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2013. cite arxiv:1312.6114.

[29] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” NIPS’14, 2014.

[30] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.

[31] F. Tramèr, A. Kurakin, N. Papernot, I. J. Goodfellow, D. Boneh, and P. D. McDaniel, “Ensemble adversarial training: Attacks and defenses,” in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[32] S. Santurkar, A. Ilyas, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, “Image synthesis with a single (robust) classifier,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 1260–1271, 2019.

[33] E. J. Martin, V. R. Polyakov, X.-W. Zhu, L. Tian, P. Mukherjee, and X. Liu, “All-assay-max2 pqsar: Activity predictions as accurate as four-concentration ic50s for 8558 novartis assays,” *Journal of Chemical Information and Modeling*, vol. 59, no. 10, pp. 4450–4459, 2019. PMID: 31518124.

[34] K. S. Sarkisyan, D. A. Bolotin, M. V. Meer, D. R. Usmanova, A. S. Mishin, G. V. Sharonov, D. N. Ivankov, N. G. Bozhanova, M. S. Baranov, O. Soylemez, N. S. Bogatyreva, P. K. Vlasov, E. S. Egorov, M. D. Logacheva, A. S. Kondrashov, D. M. Chudakov, E. V. Putintseva, I. Z. Mamedov, D. S. Tawfik, K. A. Lukyanov, and F. A. Kondrashov, “Local fitness landscape of the green fluorescent protein,” *Nature*, vol. 533, pp. 397–401, May 2016.

[35] D. H. Brookes, H. Park, and J. Listgarten, “Conditioning by adaptive sampling for robust design,” *arXiv preprint arXiv:1901.10060*, 2019.

[36] K. Hamidieh, “A data-driven statistical model for predicting the critical temperature of a superconductor,” *Computational Materials Science*, vol. 154, pp. 346 – 354, 2018.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017.

[38] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, “Stable baselines.” `https://github.com/hill-a/stable-baselines`, 2018.

[39] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *ICML*, 2018.

[40] W.-J. Shen, H.-S. Wong, Q.-W. Xiao, X. Guo, and S. Smale, “Introduction to the peptide binding problem of computational immunology: New results,” *Foundations of Computational Mathematics*, vol. 14, pp. 951–984, Oct 2014.

Appendices

In this appendix we provide additional details about the network architectures used in the ensemble of models that is used to determine when to stop taking gradient ascent steps in COMs.

A Network Details

The network architecture for each model in the ensemble consistent of an 8 layer fully connected neural network with 32 hidden unite in every layer. The number hidden units was held fixed for every task. In other words, the network architecture for these models was held fixed for every task. In addition, each hidden layer is followed by a ReLU activation function for except for a single layer, which instead uses a tanh activation function. For model i in the ensemble, every layer has a ReLU activation function except for layer i, which employs a tanh activation function. We empirically found that using an activation function that is bounded above and below, like tanh, in conjunction with an activation function that is only bounded below (like ReLU) produced the best ensembling behavior for determining when to stop taking gradient ascent steps.

B Ensemble Details

Like what was briefly mentioned in the previous section of the appendix, for every task we employ an ensemble of 8 different neural networks, using the architecture selection method described in Section A. Our selection method is intended to produce an ensemble of networks with different *architectures*, such that each model is resilient to out-of-distributions found by COMs. In practice, we found that * models is sufficient to achieve this resiliency. However, from our perspective, such an empirical trick deserves to be the subject of further investigation, and that investigation, and justification is left as future work. We stop at the max index of the min prediction in the ensemble.

C Data Collection

In this section, we detail the data collection steps used for creating each of the tasks in design-bench. We answer (1) where is the data from, and (2) what pre-processing steps are used?

C.1 GFP-v0

The GFP task provided is a derivative of the GFP dataset [34]. The dataset we use in practice is that provided by [35] at the url `https://github.com/dhbrookes/CbAS/tree/master/data`. We process the dataset such that a single training example consists of a protein represented as a tensor $x_{\text{GFP}} \in \{0, 1\}^{238 \times 20}$. This tensor is a sequence of 238 one-hot vectors corresponding to which amino acid is present in that location in the protein. We use the dataset format of [35] with no additional processing. The data was originally collected by performing laboratory experiments constructing proteins similar to the Aequorea victoria green fluorescent protein and measuring fluorescence.

C.2 MoleculeActivity-v0

The MoleculeActivity task is a derivative of a much larger dataset that is derived from ChEMBL [**?**], a large database of chemicals and their properties. The data, similar to GFP, was originally collected by performing physical experiments on a large number of molecules, and measuring their activity with a target assay. We have processed the original dataset presented in [33], which consists of more than one million molecules and 11,000 assays, into a smaller scale task with 4216 molecules and a single assay. We select this assay by first calculating the number of unique scores present in the dataset per assay, and sorting the assays by the number of unique scores. We select assay 600885 from [33]. This particular assay has 4216 molecules after pre-processing. Our pre-processing steps include converting each molecule into a one-hot tensor $x_{\text{Molecule}} \in \{0, 1\}^{1024 \times 2}$. This is performed by calculating the Morgan radius 2 substructure fingerprints of 1024 bits, which is implemented in RDKit. This calculation requires the SMILES representation for each molecule, which is provided by [33]. The final step of pre-processing, is to sub sample the dataset by defining a percentile used to select and discard high-performing molecules, such that difficulty of the task is artificially increased. We use a split percentile of 80 for MoleculeActivity in the experiments in this paper.

C.3 Superconductor-v0

Superconductor-v0 is inspired by recent work [13] that applies offline MBO to optimize the properties of superconducting materials for high critical temperature. The data we provide in our benchmark is real-world superconductivity data originally collected by [36], and subsequently made available to the public at the url `https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data#`. The original dataset consists of superconductors featurized into vectors $x_{\text{Superconductor}} \in \mathcal{R}^{81}$. One issue with the original dataset is that the largest value of a single dimension in the dataset is 22590.0, which appears to cause learning instability. We follow [13] and normalize each dimension of the design-space to have zero mean and unit variance. However, we deviate from the remaining pre-processing steps in [13]. In order to promote task realism, we directly use the superconductivity data, whereas [13] re-samples by collecting iid unit gaussian samples and labelling them with the task oracle function. This causes the scores in the dataset to correspond exactly to the scores provided by the oracle. No other domain in design-bench re-samples nor re-labels static data, so we omit it here for consistency.

C.4 HopperController-v0

The HopperController task is one that we provide ourselves. The goal of this task is to design a set of weights for as neural network policy, in order to achieve high expected return when evaluating that policy. The data collected for HopperController was taken by training a three layer neural network policy with 64 hidden units and 5126 total weights on the Hopper-v2 MuJoCo task using Proximal Policy Optimization [37]. Specifically, we use the default parameters for PPO provided in stable baselines [38]. The dataset we provide with this benchmark has 3200 unique weights. In order to collect this many, we run 32 experimental trials of PPO, where we train for one million steps, and save the weights of the policy every 10,000 environment steps. The policy weights are represented originally as a list of tensors. We first traverse this list and flatten each of the tensors, and we then concatenate each of these flattened tensors into a single training example $x_{\text{Hopper}} \in \mathcal{R}^{5126}$. The result is an optimization problem over neural network weights. After collecting these weights, we perform no additional pre-processing steps. In order to collect scores we perform a single rollout for each x using the Hopper-v2 MuJoCo environment. The horizon length for training and evaluation is limited to 1000 simulation time steps.

C.5 AntMorphology-v0 & DKittyMorphology-v0

Both morphology tasks are collected by us, and share methodology. The goal of these tasks is to design the morphology of a quadrupedal robot—an ant or a D'Kitty—such that the agent is able to crawl quickly in a particular direction. In order to collect data for this environment, we create variants of the MuJoCo Ant and the ROBEL D'Kitty agents that have parametric morphologies. The goal is to determine a mapping from the morphology of the agent to the average return that an agent trained for a particular intended morphology achieves. We implement this by pre-training a neural network policy using SAC [39]. For both the Ant and the D'Kitty, we train agents for up to three million environments steps, and a maximum episode length of 1000, with all other settings as default. These agents are pre-trained for a fixed gold-standard morphology—the default morphology of the Ant and D'Kitty respectively. Each morphology task consists of samples obtained by adding Gaussian noise with standard deviation 0.02 for Ant and 0.01 for DKitty times the design-space range to the gold-standard morphology. We label each sampled morphology by averaging the return of 16 rollouts of length 100 of an agent with that morphology.

D Oracle Functions

We detail oracle functions for evaluating ground truth scores for each of the tasks in design-bench. A common thread among these is that the oracle, if trained, is fit to a larger static dataset containing higher performing designs than observed by a downstream MBO algorithm.

D.1 GFP-v0

GFP-v0 uses the oracle function from [35]. This oracle is a Gaussian Process regression model with a protein-specific kernel proposed by [40]. The Gaussian Process is fit to a larger dataset than the static dataset packaged with GFP-v0, making it possible to sample a protein design that achieve a higher score than any other protein seen during training. The oracle score for GFP-v0 is implemented as the mean prediction of this Gaussian Process.

D.2 MoleculeActivity-v0

Following the procedure set by [33], the oracle function we use for MoleculeActivity-v0 is a random forest regression model. In particular, we use the RandomForestRegressor provided in scikit-learn, using identical hyperparameters to the random forest regression model used in [33]. The random forest is trained on the entire task dataset. In practice, samples that score at most the 80th percentile are observed by an MBO algorithm, which allows for sampling *unobserved* points that score higher than the highest training point.

D.3 Superconductor-v0

The Superconductor-v0 oracle function is also a random forest regression model. The model we use it the model described by [36]. We borrow the hyperparameters described by them, and we use the RandomForestRegressor provided in scikit-learn. Similar to the setup for the previous two tasks, this oracle is trained on the entire static dataset, and the task is instantiated with a split percentile. Samples scoring at most in the 80th percentile are observed by an MBO algorithm, which allows for sampling *unobserved* points that score in the unobserved top 20 percent.

D.4 HopperController-v0

Unlike the previously described tasks, HopperController-v0 and the remaining tasks implement an exact oracle function. For HopperController-v0 the oracle takes the form of a single rollout using the Hopper-v2 MuJoCo environment. The designs for HopperController-v0 are neural network weights, and during evaluation, a policy with those weights is instantiated—in this case that policy is a three layer neural network with 11 input units, two layers with 64 hidden units, and a final layer with 3 output units. The intermediate activations between layers are hyperbolic tangents. After building a policy, the Hopper-v2 environment is reset and the reward for 1000 time-steps is summed. That summed reward constitutes the score returned by the HopperController-v0 oracle. The limit of performance is the maximum return that an agent can achieve in Hopper-v2 over 1000 steps.

D.5 AntMorphology-v0 & DKittyMorphology-v0

The final two tasks in design-bench use an exact oracle function, using the MuJoCo simulator. For both morphology tasks, the simulator performs 16 rollouts and averages the sum of rewards attained over them. Each task is accompanied by a pre-trained neural network policy. To perform evaluation, a morphology is passed to the Ant or D'Kitty MuJoCo environments respectively, and a dynamic-morphology agent is initialized inside these environments. These environments are very sensitive to small morphological changes, and exhibit a high degree of stochasticity as a result. To compensate for the increased stochasticity, we average returns over 16 rollouts.