Conservative Objective Models for Effective Offline Model-Based Optimization

Brandon Trabucco [*1] **Aviral Kumar** [*1] **Xinyang Geng** [1] **Sergey Levine** [1]

Abstract

Computational design problems arise in a number of settings, from synthetic biology to computer architectures. In this paper, we aim to solve data-driven model-based optimization (MBO) problems, where the goal is to find a design input that maximizes an unknown objective function provided access to only a static dataset of prior experiments. Such data-driven optimization procedures are the only practical methods in many real-world 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 design against a learned model suffer from distributional shift: it is easy to find a design that "fools" the model into predicting a high value. To overcome this, we propose *conservative objective models* (COMs), a method that learns a model of the objective function that lower bounds the actual value of the ground-truth objective on out-of-distribution inputs, and uses it for optimization. Structurally, COMs resembles adversarial training methods used to overcome adversarial examples. COMs are simple to implement, and outperform a number existing methods on a wide range of MBO problems, including optimizing protein sequences, robot morphologies, neural network weights, and superconducting materials.

1. Introduction

Black-box model-based optimization (MBO) problems are ubiquitous in a wide range of domains, such as protein (Brookes et al., 2019) or molecule design (Gaulton et al., 2012), designing controllers (Berkenkamp et al., 2016) or robot morphologies (Liao et al., 2019), optimizing neural network designs (Zoph & Le, 2017), and aircraft design (Hoburg & Abbeel, 2012). Existing methods to solve such model-based optimization problems typically learn a proxy function to represent the unknown objective landscape based on the data, and then optimize the design against this learned objective function. In order to prevent errors in the learned proxy function from affecting optimization, these methods often critically rely on periodic *active* data collection (Snoek et al., 2012) over the course of training. Active data collection can be expensive or even dangerous: evaluating a real design might involve a complex real-world procedure such as synthesizing candidate protein structures for protein optimization or building the robot for robot design optimization. While these problems can potentially be solved via computer simulation, a high fidelity simulator often requires considerable effort from experts across multiple domains to build, making it impractical for most problems. Therefore, a desirable alternative approach for a broad range of MBO problems is to develop *data-driven, offline* methods that can optimize designs by training highly general and expressive deep neural network models on data from previously conducted experiments, consisting of inputs ($\mathbf{x}$) and their corresponding objective values (y), without access to the true function or any form of active data collection (Kumar & Levine, 2019). In a number of these practical domains, such as protein (Sarkisyan et al., 2016a) or molecule design (Gaulton et al., 2012), plenty of prior

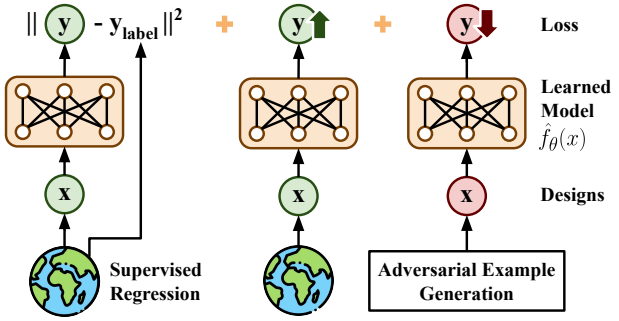

Figure 1. **Overview of COMs.** Our method trains a model of the objective function by training a neural net with supervised regression on the training data augmented two additional loss terms to obtian conservative predictions. These additional terms aim to maximize the predictions of the neural net model on the training data, and minimize the predictions on adversarially generated designs. This principle prevents the optimizer from producing bad designs with erroneously high values at unseen and poor designs.

[*]Equal contribution [1]Department of Electrical Engineering and Computer Sciences, University of California Berkeley. Correspondence to: Brandon Trabucco <btrabucco@berkeley.edu>, Aviral Kumar <aviralk@berkeley.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

data already exists and can be utilized for fully offline, data-driven model-based optimization.

Typical approaches for addressing MBO problems learn a model of the unknown objective function $\hat{f}$ that maps an input $\mathbf{x}$ (or a representation of the input (Gómez-Bombarelli et al., 2018)) to its objective value $\hat{f}(\mathbf{x})$ via supervised regression on the training dataset (Snoek et al., 2012). Then, these methods 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 erroneously drive the optimization procedure towards out-of-distribution, invalid, and low-valued design designs that "fool" the model into producing a high values (Kumar & Levine, 2019).

How can we prevent offline MBO methods from falling into such out-of-distribution solutions? If we can instead learn a *conservative* model of the objective function that does not overestimate the objective value on out-of-distribution inputs, optimizing against this conservative model would produce the best solutions for which we are *confident* in the value. In this paper, we propose a method to learn such *conservative objective models* (COMs), and then optimize the design against this conservative model using a naïve gradient-ascent procedure. Analogously to adversarial training approaches in supervised learning (Goodfellow et al., 2014b), and building on recent works in offline reinforcmeent learning (Levine et al., 2020; Kumar et al., 2020), COMs first explicitly mine for out-of-distribution inputs with erroneously overestimated values and then penalize the predictions on these inputs. Theoretically, we show that this approach mitigates overestimation in the learned objective model near the manifold of the dataset. Empirically, we find that this leads to good performance across a range of offline model-based optimization tasks.

The primary contribution of this paper, COMs, is a novel approach for addressing data-driven model-based optimization problems by learning a conservative model of the unknown objective function that lower-bounds the groundtruth function on out-of-distribution inputs, and then optimizing the input against this conservative model via a simple gradient-ascent style procedure. COMs are simple to implement, utilizing a supervised learning procedure that resembles adversarial training, without the need for complex generative modeling to estimate dataset support as in prior work on model-based optimization. We theoretically analyze COMs and show that they never overestimate the values at out-of-distribution inputs close to the dataset manifold and we empirically demonstrate the efficacy of COMs on seven complex MBO tasks that span a wide range of real-world tasks including biological sequence design, neural network parameter optimization, and superconducting material design. COMs is optimal on **4/7** tasks, and outperforms the best prior method by a factor of **1.3x** in a high-dimensional setting, and by a factor of **1.16x** overall.

2. Preliminaries

The goal in data-driven, offline model-based optimization (Kumar & Levine, 2019) is to find the best possible 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. An offline 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 (Gómez-Bombarelli et al., 2018; Brookes et al., 2019; Kumar & Levine, 2019; Fannjiang & Listgarten, 2020), 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. One choice for optimizing $\mathbf{x}$ in Equation 1 is gradient descent on the learned function, as given by

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

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 thin 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. Prior methods have sought to address this issue via generative modeling or explicit density estimation, so as to avoid out-of-distribution inputs. In the next section, we will describe how our method, COMs, instead trains the objective model in such a way that overestimation is prevented directly.

3. Conservative Objective Models for Offline Model-Based Optimization

In this section, we present our approach, conservative objective models (COMs). COMs learn estimates of the true function that do not overestimate the value of the ground truth objective on out-of distribution inputs in the vicinity of the training dataset. As a result, COMs prevent erroneous overestimation that would drive the optimizer (Equation 2) to produce out-of-distribution inputs with low values under the groundtruth objective function. We first discuss a procedure for learning such conservative estimates and

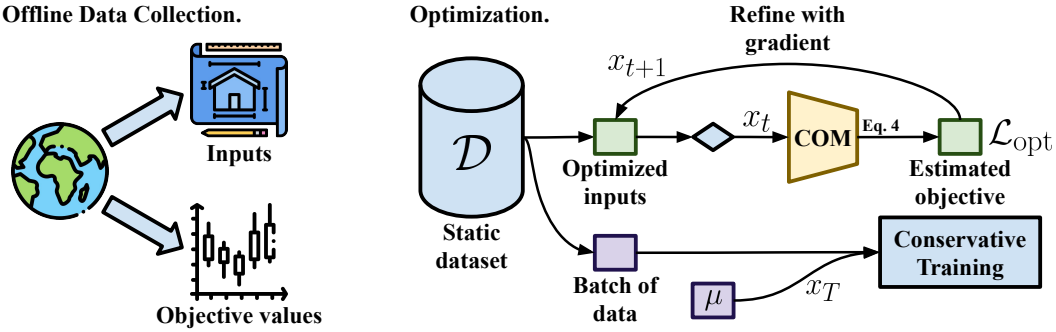

Figure 2. **Training and optimization using COMs.** The section on the left indicates that each task provides a static dataset that is collected offline without ayn MBO algorithm in-the-loop. The section on the right shows how a conservative objective model is used to produce promising optimized designs using gradient ascent, and how these designs are inputs to a conservative regularizer.

then explain how these conservative models can be used for offline MBO.

3.1. Learning Conservative Objective Models (COMs)

The key idea behind our approach is to augment the objective for training of the objective model, $\hat{f}_\theta(\mathbf{x})$, with a regularizer that minimizes the expected value of this function on "adversarial" inputs where the value of the learned function $\hat{f}_\theta$ may be erroneously large. Such adversarial inputs are likely to be found by the optimizer during optimization, and hence, we need to train the learned function to not overestimate their values. How can we compute such adversarial inputs? Building on simple techniques for generating adversarial examples in supervised learning (Goodfellow et al., 2014b), we can run multiple steps of gradient ascent on the current snapshot of the learned function $\hat{f}(\mathbf{x})$ starting from various inputs in the training dataset to obtain such adversarial inputs. For concise notation in the exposition, we denote the distribution of all adversarial inputs found via this gradient ascent procedure as $\mu(\mathbf{x})$. Samples from $\mu(\mathbf{x})$ are obtained by sampling a datapoint from the training set and running several steps of gradient ascent on $\hat{f}(\mathbf{x})$.

$$\mu(\mathbf{x}) = \sum_{\mathbf{x}_0 \in \mathcal{D}} \delta_{\mathbf{x}=\mathbf{x}_T} : \mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla_{\mathbf{x}} \hat{f}_\theta(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_t} \quad (3)$$

While simply minimizing the function values under this adversarial distribution $\mu(\mathbf{x})$ should effectively reduce the value of the learned $\hat{f}$ at these inputs, this can result in systematic underestimation even for in-distribution points. To balance out this regluarization, our approach additionally *maximizes* the expected value of this function on the training dataset. This can be formalized as maximizing the value of $\hat{f}(\mathbf{x})$ under the empirical distribution of inputs $\mathbf{x} \in \mathcal{D}$ given by: $\hat{D}(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{D}} \delta_{\mathbf{x}=\mathbf{x}_i}$. In Section 4, we will show that the minimization and maximization terms balance out, and this objective learns a function $\hat{f}_\theta(\mathbf{x})$ that is a lower bound on the true function $f(\mathbf{x})$ for inputs that are encountered during the optimization process, under several assumptions. This approach is inspired by recent work in offline RL (Kumar et al., 2020), where a similar objective is used to learn conservative value functions. We will elaborate on this connection in Section 5. Formally, our training objective is given by the following equation, where α is a parameter that trades off conservatism for regression:

$$\hat{f}_\theta^\star \leftarrow \arg\min_{\theta \in \Theta} \overbrace{\alpha \left(\mathbb{E}_{\mathbf{x} \sim \mu(\mathbf{x})} \left[\hat{f}_\theta(\mathbf{x}) \right] - \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\hat{f}_\theta(\mathbf{x}) \right] \right)}^{\text{COMs regularizer}} + \underbrace{\frac{1}{2} \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} \left[\left(\hat{f}_\theta(\mathbf{x}) - y \right)^2 \right]}_{\text{standard supervised regression}}, \quad (4)$$

This idea is schematically depicted in Figure 1. The value of α and the choice of distribution $\mu(\mathbf{x})$ play a crucial role in determining the behavior of this approach. If the chosen α is very small, then the resulting $\hat{f}_\theta^\star(\mathbf{x})$ may not be a conservative estimate of the actual function $f(\mathbf{x})$, whereas if the chosen α is too large, then the learned function will be too conservative, and not allow the optimizer to deviate away from the dataset at all. We will discuss our strategy for choosing α in the next section. As noted earlier, our choice of $\mu(\mathbf{x})$ specifically focuses on adversarial inputs that the optimizer is likely to encounter while optimizing the input. We compute this distribution $\mu(\mathbf{x})$ by sampling a starting point $\mathbf{x}_0$ from the dataset $\mathcal{D}$, and then performing several steps of gradient ascent on $\hat{f}_\theta$ starting from this point.

3.2. Optimizing a Conservative Objective Model

Once we have a trained conservative model from Equation 4, we must use this learned model for finding the best possible input, $\mathbf{x}^\star$. Prior works (Kumar & Levine, 2019; Brookes et al., 2019) use a standard (non-conservative) model of the objective function in conjunction with generative models or density estimators to restrict the optimization to in-distribution values of $\mathbf{x}^\star$. However, since our conservative training method trains $\hat{f}_\theta^\star$ to explicitly assign low values

Algorithm 1 COM: Training Conservative Models

1: Initialize $\hat{f}_\theta$. Pick η, α and initialize dataset $\mathcal{D}$.
2: **for** $i = 1$ to training_steps **do**
3: Sample $(\mathbf{x}_0, y) \sim \mathcal{D}$
4: Find $\mathbf{x}_T(\mathbf{x}_0)$ via gradient ascent from $\mathbf{x}_0$:
$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla_{\mathbf{x}} \hat{f}_\theta(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_t}$; $\mu(\mathbf{x}) = \sum_{\mathbf{x}_0 \in \mathcal{D}} \delta_{\mathbf{x}=\mathbf{x}_T(\mathbf{x}_0)}$.
5: Minimize $\mathcal{L}(\theta; \alpha)$ with respect to θ.
$\mathcal{L}(\theta;\alpha) = \mathbb{E}_{\mathbf{x}_0 \sim \mathcal{D}}(\hat{f}_\theta(\mathbf{x}_0) - y)^2 - \alpha \mathbb{E}_{\mathbf{x}_0}[\hat{f}_\theta(\mathbf{x}_0)] + \alpha \mathbb{E}_{\mu(\mathbf{x})}[\hat{f}_\theta(\mathbf{x})]$
$\theta \leftarrow \theta - \lambda \nabla_\theta \mathcal{L}(\theta; \alpha)$
6: **end for**

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

1: Initialize optimizer at the optimum in $\mathcal{D}$:
$\tilde{\mathbf{x}} = \arg\max_{(\mathbf{x},y) \in \mathcal{D}} y$
2: Find $\mathbf{x}^\star$ via gradient ascent from $\tilde{\mathbf{x}}$:
$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla_{\mathbf{x}} \mathcal{L}_{\text{opt}}(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_t}$
where $\mathcal{L}_{\text{opt}}(\mathbf{x}) := \hat{f}^\star_\theta(\mathbf{x})$
3: Return the solution $\mathbf{x}^\star = \mathbf{x}_T$.

to out-of-distribution inputs, we can use a simple gradient-ascent style procedure in the input space to find the best possible solution.

Specifically, our optimizer runs gradient-ascent for T iterations starting from an input in the dataset ($\mathbf{x}_0 \in \mathcal{D}$), in each iteration trying to move the design in the direction of the gradient of the learned model $\hat{f}^\star_\theta$. Starting from the best point in the dataset, $\mathbf{x}_0 \in \mathcal{D}$, our optimizer performs the following update (also shown in Algorithm 2, Line 2):

$$\forall\, t \in [T], \mathbf{x}_0 \in \mathcal{D}; \quad \mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla_{\mathbf{x}} \mathcal{L}_{\text{opt}}(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_t}$$
$$\text{where } \mathcal{L}_{\text{opt}}(\mathbf{x}) := \hat{f}^\star_\theta(\mathbf{x}). \tag{5}$$

Equation 5 ensures that the value of the learned function $\hat{f}_\theta(\mathbf{x}_{t+1})$ is larger than the value at its previous iterate $\mathbf{x}_t$. Furthermore, the number of iterations T of gradient ascent during optimization in Equation 5 is the identical to the number of steps that we use to generate adversarial examples, $\mu(\mathbf{x})$ in Equation 3. This ensures that the optimizer only queries the region when the learned function $\hat{f}_\theta(\mathbf{x})$ is indeed conservative and a valid lower bound.

3.3. Using COMs for MBO: Additional Decisions

Next we discuss other design decisions that appear in COMs training (Equation 4) or when optimizing the input against a learned conservative model (Equation 5).

Choosing α. The hyperparameter α in Equation 4 plays an important role in weighting conservatism against accuracy. Without access to additional active data collection for evaluation, tuning this hyperparameter for each task can be challenging. Therefore, in order to turn COMs into a task-agnostic algorithm for offline MBO, we devise an automated procedure for selecting α. As discussed previously, if α is too large, $\hat{f}^\star_\theta$ is expected to be too conservative, since it would assign higher values to points in the dataset, and low values to *all* other points. Selecting a single value of α that works for many problems is difficult, since its effect depends strongly on the magnitude of the objective function. Instead, we use a modified training procedure that poses Equation 4 as a constrained optimization problem, with α assuming the role of a Lagrange dual variable for satisfying a constraint that controls the difference in values of the learned objective under $\mu(\mathbf{x})$ and $\mathcal{D}(\mathbf{x})$. This corresponds to solving the following optimization problem:

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

While Equation 6 introduces a new hyperparameter τ in place of α, this parameter is easier to select by hand, since its optimal value does not depend on the magnitude of the objective function as we can normalize the objective values to the same range before use in Equation 6, and therefore, a single choice works well across a diverse range of tasks. We find that a single value of τ is effective on every continuous task ($\tau = 0.5$) and discrete task ($\tau = 2.0$) respectively, and empirically ablate the choice of τ in Figure 3.

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 5) encountered in the optimization trajectory should be used as our final solution $\mathbf{x}^\star$. The most natural choice is to pick the final $\mathbf{x}_T$ found by the optimizer as the solution. We uniformly choose $T = 50$ steps. While the choice of T should, in principle, affect the solution found by any gradient-ascent style optimizer, we found COMs to be quite stable to different values of T, as we will elaborate empirically on in Section 6.2, Figure 3. Of course, there are many other possible ways of selecting T, including ideas inspired from offline model-selection methods in offline reinforcement learning (Thomas et al., 2015), but our simple procedure, which is also popular in offline RL (Fu et al., 2020), ensures that the optimizer only queries the regions of the input space where the learned function is indeed trained to be conservative and is also sufficient to obtain good optimization performance.

3.4. Overall Algorithm and Practical Implementation

Finally, we combine the individual components discussed so far to obtain a complete algorithm for offline model-based optimization. Pseudocode for our algorithm is shown in Al-

gorithm 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 $\mu(\mathbf{x})$ via gradient ascent (Line 4), and optimizing parameters θ using Equation 5 (Line 5). 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 number of ascent steps and return the final design $\mathbf{x}_T$ as $\mathbf{x}^\star$.

Implementation details. Full implementation details for our method can be found in Appendix A. Briefly, 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 details on the network structure can be found in Appendix C. In order to train this conservative objective model, we use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 10^{-3}. Empirically, we found that if η is too large, gradient ascent begins to produce inputs $\mathbf{x}_T$ that do not maximize the values of $\hat{f}^\star_\theta(\mathbf{x}_T)$, so we select the largest η such that successive $\mathbf{x}_t$ follow the gradient vector field of $\hat{f}^\star_\theta(\mathbf{x}_t)$. For computing samples $\mu(\mathbf{x})$, we used 50 gradient ascent steps starting from a given design in the dataset, $\mathbf{x}_0 \in \mathcal{D}$. During optimization, we used the gradient-ascent optimizer with a learning rate of 0.05 for continuous tasks and 2.0 for discrete tasks. As we will also show in our experiments (Section 6.2), this produces stable optimization behavior for all tasks we attempted. Finally, in order to choose the step T in Equation 5 that is supposed to provide us with the final solution $\mathbf{x}^\star = \mathbf{x}_T$, we pick a universal step of $T = 50$.

4. Theoretical Analysis of COMs

We will now theoretically analyze conservative objective models, and show that the conservative training procedure (Equation 4) indeed learns a conservative model of the objective function. To do so, we will show that under Equation 4, the values of all inputs in regions found within T steps of gradient ascent starting from any input $\mathbf{x}_0 \in \mathcal{D}$ are lower-bounds on their actual value. For analysis, we will denote $\overline{\mathcal{D}}(\mathbf{x})$ as the smoothed density of $\mathbf{x}$ in the dataset $\mathcal{D}$ (see Appendix B for a formal definition). We will express Equation 4 in an equivalent form that factorizes the distribution $\mu(\mathbf{x})$ as $\mu(\mathbf{x}) = \sum_{\mathbf{x}_0 \sim \mathcal{D}} \overline{\mathcal{D}}(\mathbf{x}_0)\mu(\mathbf{x}_T|\mathbf{x}_0)$:

$$\min_{\theta \in \Theta} \alpha \left(\mathbb{E}_{\mathbf{x}_0 \sim \overline{\mathcal{D}}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)} \left[\hat{f}_\theta(\mathbf{x}_T) \right] - \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}} \left[\hat{f}_\theta(\mathbf{x}) \right] \right) + \frac{1}{2} \mathbb{E}_{(\mathbf{x}, y) \sim \overline{\mathcal{D}}} \left[\left(\hat{f}_\theta(\mathbf{x}) - y \right)^2 \right]. \quad (7)$$

While $\mu(\mathbf{x}_T|\mathbf{x}_0)$ is a Dirac-delta distribution in practice (Section 3), for our analysis, we will assume that it is a distribution centered at $\mathbf{x}_T$ and $\mu(\mathbf{x}_T|\mathbf{x}_0) > 0 \; \forall \; \mathbf{x}_T \in \mathcal{X}$. This condition can be easily satisfied by adding random noise during gradient ascent while computing $\mathbf{x}_T$. We will train $\hat{f}_\theta$ using gradient descent and denote $k = 1, 2, \cdots$ as the iterations of this training procedure for $\hat{f}_\theta$.

We first summarize some assumptions used in our analysis. We assume that the true function $f(\mathbf{x})$ is L-Lipschitz over the input space $\mathbf{x}$. We also assume that the learned function $\hat{f}_\theta(\mathbf{x})$ is $\widehat{L}$-Lipschitz and $\widehat{L}$ is sufficiently larger than L. For analysis purposes, we will define a conditional distribution, $\overline{\mathcal{D}}(\mathbf{x}'|\mathbf{x})$, to be a Gaussian distribution centered at $\mathbf{x}$: $\mathcal{N}(\mathbf{x}'|\mathbf{x}, \sigma^2)$. We will not assume a specific parameterization for the objective model, $\hat{f}_\theta$, but operate under the neural tangent kernel (NTK) (Jacot et al., 2018) model of neural nets. The neural tangent kernel of the function $\hat{f}(\mathbf{x})$ be defined as: $\mathbf{G}_f(\mathbf{x}_i, \mathbf{x}_j) := \nabla_\theta \hat{f}_\theta(\mathbf{x}_i)^T \nabla_\theta \hat{f}_\theta(\mathbf{x}_j)$. Under these assumptions, we build on the analysis of conservative Q-learning (Kumar et al., 2020) to prove our theoretical result in Theorem 1, shown below:

Proposition 1 (Conservative training lower-bounds the true function)**.** *Assume that $\hat{f}_\theta(\mathbf{x})$ is trained with conservative training by performing gradient descent on θ with respect to the objective in Equation 7 with a learning rate η. The parameters in step k of gradient descent are denoted by θ^k, and let the corresponding conservative model be denoted as $\hat{f}^k_\theta$. Let $\mathbf{G}$, μ, $\widehat{L}$, L, $\overline{\mathcal{D}}$ be defined as discussed above. Then, under assumptions listed above, $\forall \; \mathbf{x} \in \mathcal{D}, \mathbf{x}'' \in \mathcal{X}$, the conservative model at iteration $k+1$ of training satisfies:*

$$\hat{f}^{k+1}_\theta(\mathbf{x}'') := \max \Big\{ \hat{f}^{k+1}_\theta(\mathbf{x}) - \widehat{L}||\mathbf{x}'' - \mathbf{x}||_2, \; \tilde{f}^{k+1}_\theta(\mathbf{x}'') - \eta\alpha \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}, \mathbf{x}' \sim \mu}[\mathbf{G}^k_f(\mathbf{x}'', \mathbf{x}')] + \eta\alpha \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}, \mathbf{x}' \sim \overline{\mathcal{D}}}[\mathbf{G}^k_f(\mathbf{x}'', \mathbf{x}')] \Big\},$$

where $\tilde{f}^{k+1}_\theta(\mathbf{x}'')$ is the resulting $(k+1)$-th iterate of $\hat{f}_\theta$ if conservative training were not used. Thus, if α is sufficiently large, the expected value of the asymptotic function, $\hat{f}_\theta := \lim_{k \to \infty} \hat{f}^k_\theta$, on inputs $\mathbf{x}_T$ found by the optimizer, lower-bounds the value of the true function $f(\mathbf{x}_T)$:

$$\mathbb{E}_{\mathbf{x}_0 \sim \mathcal{D}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[\hat{f}_\theta(\mathbf{x}_T)] \leq \mathbb{E}_{\mathbf{x}_0 \sim \mathcal{D}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[f(\mathbf{x})].$$

A proof for Proposition 1 including a complete formal statement can be found in Appendix B. The intuition behind the proof is that inducing conservatism in the function $\hat{f}_\theta$ at each gradient step of optimizing Equation 7 makes the asymptotic function be conservative. Moreover, the larger the value of α, the more conservative the function $\hat{f}_\theta$ is on points $\mathbf{x}'$ found via gradient ascent, i.e., points with high density under $\mu(\mathbf{x}_T|\mathbf{x}_0)$, in expectation. Finally, when gradient ascent is used to find $\mathbf{x}^\star$ on the learned conservative model, $\hat{f}_\theta$, and the number of steps of gradient ascent steps is less than T, as we do in practice via Equation 5, this bound with additional offset will hold for the point $\mathbf{x}^\star$ in expectation, and therefore the estimated value of this point

will not overestimate its true value. This additional offset depends on the Lipschitz constant $\widehat{L}$ and the distance between $\mathbf{x}^*$ and the the optimized solutions $\mathbf{x}_T$ found for other data points, $\mathbf{x}_0 \in \mathcal{D}$.

5. Related Work

We now briefly discuss prior works in MBO, 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 (Rubinstein & Kroese, 2004) and other methods derived from the REINFORCE trick (Williams, 1992b; Rubinstein, 1996), reward-weighted regression (Peters & Schaal, 2007), and Gaussian processes (Snoek et al., 2015; Shahriari et al., 2016; Snoek et al., 2012) 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 (Snoek et al., 2015; 2012), latent variable models (Kim et al., 2019; Garnelo et al., 2018b;a), and ensembles of learned score models (Angermueller et al., 2020a;b; Mirhoseini et al., 2020). These methods still require actively querying the true function $f(\mathbf{x})$. Further, as shown by (Brookes et al., 2019; Fannjiang & Listgarten, 2020; Kumar & Levine, 2019), 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 generative model (Kingma & Welling, 2013; Goodfellow et al., 2014a) that models the manifold of inputs. (Brookes et al., 2019; Fannjiang & Listgarten, 2020) use a variational autoencoder (Kingma & Welling, 2013) to model the space of $\mathbf{x}$ and use it alongside a learned objective function. (Kumar & Levine, 2019) use a generative model to parameterize 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 6), and as a result these generative models often need to be tuned for each domain (Trabucco et al., 2021). In contrast, COMs do not require any generative model, and fit an approximate objective function with a simple regularizer, providing both a simpler, easier-to-use algorithm and better empirical performance. Fu & Levine (2021) also avoid training a generative model, but instead use normalized maximum likelihood, which requires training multiple discriminative models—COMs only requires one—and quantizing y, which COMs does not.

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 (Goodfellow et al., 2014b), which can be generated by maximizing the input against the loss function. While 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 (Tramèr et al., 2018) of the learned function, and there is evidence that robust objective models help mitigate over estimation (Santurkar et al., 2019), 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.

6. Experimental Evaluation

To evaluate the efficacy of COMs for offline model-based optimization, we first perform a comparative evaluation of COMs on four continuous and three discrete offline MBO tasks based on problems in physical sciences, neural network design, material design, and robotics, proposed in the design-bench benchmark (Trabucco et al., 2021), that we also describe shortly. In addition, we perform an empirical analysis on COMs that aims to answer the following questions: **(1)** Is conservative training essential for improved performance and stability of COMs? How do COMs compare to a naïve objective model in terms of stability?, **(2)** How sensitive are COMs are to various design choices during optimization?, **(3)** Are COMs robust to hyperparameter choices and consistent to evaluation conditions? We answer these questions by studying the behavior of COMs under controlled conditions and using visualizations for our analysis. Code for reproducing our results is at https://github.com/brandontrabucco/design-baselines

6.1. 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 (Brookes et al., 2019), MINs (Kumar & Levine, 2019) and and autofocused CbAS (Fannjiang & Listgarten, 2020), that augments CbAS with a re-weighted objective model. Additionally, we also compare COMs to more standard baseline algorithms including REINFORCE (Williams, 1992a), CMA-ES (Hansen, 2006) , and BO-qEI, Bayesian Optimization with the quasi-expected improvement acquisition function (Wilson et al., 2017). We also compare to a naïve **gradient ascent** baseline that first learns a model of the

	GFP	TF Bind 8	UTR	# Optimal	Norm. avg. perf.
$\mathcal{D}$ **(best)**	0.789	0.439	0.593		
Auto. CbAS	**0.865 ± 0.000**	0.910 ± 0.044	0.691 ± 0.012	1 / 7	0.687
CbAS	**0.865 ± 0.000**	0.927 ± 0.051	**0.694 ± 0.010**	3 / 7	0.699
BO-qEI	0.254 ± 0.352	0.798 ± 0.083	0.684 ± 0.000	0 / 7	0.629
CMA-ES	0.054 ± 0.002	0.953 ± 0.022	**0.707 ± 0.014**	2 / 7	0.674
Grad.	0.864 ± 0.001	**0.977 ± 0.025**	**0.695 ± 0.013**	3 / 7	0.750
Grad. Min	0.864 ± 0.000	**0.984 ± 0.012**	**0.696 ± 0.009**	3 / 7	0.829
Grad. Mean	0.864 ± 0.000	**0.986 ± 0.012**	0.693 ± 0.010	2 / 7	0.852
MINs	**0.865 ± 0.001**	0.905 ± 0.052	**0.697 ± 0.010**	**4 / 7**	0.745
REINFORCE	**0.865 ± 0.000**	0.948 ± 0.028	0.688 ± 0.010	1 / 7	0.541
COMs (Ours)	*0.864 ± 0.000*	*0.945 ± 0.033*	**0.699 ± 0.011**	**4 / 7**	**0.985**

	Superconductor	Ant Morphology	D'Kitty Morphology	Hopper Controller	
$\mathcal{D}$ **(best)**	0.399	0.565	0.884	1.0	
Auto. CbAS	0.421 ± 0.045	0.882 ± 0.045	0.906 ± 0.006	0.137 ± 0.005	
CbAS	**0.503 ± 0.069**	0.876 ± 0.031	0.892 ± 0.008	0.141 ± 0.012	
BO-qEI	0.402 ± 0.034	0.819 ± 0.000	0.896 ± 0.000	0.550 ± 0.118	
CMA-ES	0.465 ± 0.024	**1.214 ± 0.732**	0.724 ± 0.001	0.604 ± 0.215	
Grad.	**0.518 ± 0.024**	0.293 ± 0.023	0.874 ± 0.022	1.035 ± 0.482	
Grad. Min	**0.506 ± 0.009**	0.479 ± 0.064	0.889 ± 0.011	1.391 ± 0.589	
Grad. Mean	**0.499 ± 0.017**	0.445 ± 0.080	0.892 ± 0.011	1.586 ± 0.454	
MINs	0.469 ± 0.023	**0.913 ± 0.036**	**0.945 ± 0.012**	0.424 ± 0.166	
REINFORCE	0.481 ± 0.013	0.266 ± 0.032	0.562 ± 0.196	-0.020 ± 0.067	
COMs (Ours)	0.439 ± 0.033	**0.944 ± 0.016**	**0.949 ± 0.015**	**2.056 ± 0.314**	

Table 1. **Comparative evaluation of COMs** against prior methods in terms of the mean 100th-percentile score and its standard deviation over 8 trials. Tasks include **Superconductor-RandomForest-v0**, **HopperController-Exact-v0**, **AntMorphology-Exact-v0**, and **DKittyMorphology-Exact-v0**, which have a continuous design space and **GFP-Transformer-v0**, **TFBind8-Exact-v0**, and **UTR-ResNet-v0** with a discrete design space. COMs perform strictly better on high-dimensional tasks, obtaining about **1.3x** gains on Hopper Controller, and compelling gains on Ant Morphology and D'Kitty Morphology tasks. In addition, COMs is able to consistently find solutions that outperform the best training point for each task, given by $\mathcal{D}$ **(best)**. For each task, algorithms within one standard deviation of having the highest performance are bolded. COMs attain the optimal performance in 4/7 tasks ("# Optimal") attaining a normalized average performance of **0.985** compared to 0.852 for the next best method, outperforming other methods as indicated.

actual function via supervised regression (with no conservative term like COMs) and then optimizes this learned proxy via gradient ascent. CbAS variants and MINs train generative models such as VAEs (Kingma & Welling, 2013) and GANs (Goodfellow et al., 2014a), which generally require task-specific neural net architectures, as compared to the substantially simpler discriminative models used for COMs. In fact, we use the same architecture for COMs on all the tasks. In addition, we instantiate this gradient ascent baseline with an ensemble of learned models of the objective function, with either a minimum (Grad. Min.) or mean (Grad. Mean) over the ensemble to obtain a learned prediction that is then optimized via gradient ascent.

Evaluation protocol. Our evaluation protocol follows prior work (Brookes et al., 2019; Trabucco et al., 2021): we query each method to obtain the top $N = 128$ most promising optimized samples $\mathbf{x}_1^\star, \cdots, \mathbf{x}_N^\star$ according to the model, and then report the 100^{th} percentile ground truth objective values on this set of samples, $\max(\mathbf{x}_1^\star, \cdots, \mathbf{x}_N^\star)$, as well as the 50^{th} percentile objective values (See Appendix A for numbers), averaged over 8 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 deployment.

Offline MBO tasks. The tasks we use can be found in the design-bench benchmark (Trabucco et al., 2021) at github.com/brandontrabucco/design-bench. Here we briefly summarize the tasks: **(A)** Superconductor (Fannjiang & Listgarten, 2020), where the goal is to optimize over 86-dimensional superconductor designs to maximize the critical temperature using 21263 points, **(B)** Hopper Controller (Kumar & Levine, 2019), where the goal is to optimize over 5126-dimensional weights of a neural network policy on

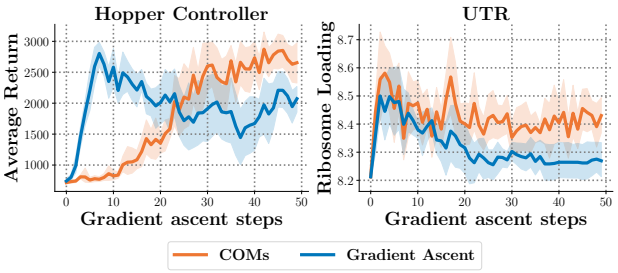

Figure 3. **Stability of COMs versus naïve gradient ascent.** The x-axis shows the number of gradient ascent steps taken on the design $\mathbf{x}^*$, and the y-axis shows the 100th percentile of the ground truth task objective function evaluated at every gradient step, which is used only for analysis only and is unavailable to the algorithm. In both cases, COMs reach solutions that remain at higher performance stably, indicating that COMs are less sensitive to varying numbers of gradient ascent steps performed during optimization.

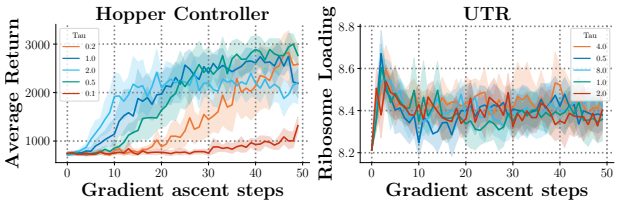

Figure 4. **Ablation of stability and universality of τ.** In each of the two plots, we instantiate COMs on the HopperController and UTR tasks, and vary τ that controls the degree of conservatism (Equation 6). The x-axis denotes the number of gradient ascent steps taken on the design $\mathbf{x}^*$ with respect to $\hat{f}_\theta$, and the y-axis indicates the 100th percentile of the ground truth function $\mathbf{x}$, which remains unobserved by the COMs algorithm, and only serves as an ablative visualization. The results demonstrate that increasing τ improves stability of COMs, and that COMs is robust to the particular choice of τ. We select $\tau = 0.5$ universally for continuous tasks, and $\tau = 2.0$ universally for discrete tasks.

the Hopper-v2 gym domain using a dataset of 3200 points, and **(C)** Ant and **(D)** D'Kitty Morphology, where the goal is to design the 60 and 56-dimensional morphologies, respectively, of robots to maximize policy performance using datasets, both of size 25009. We also evaluate COMs on tasks with a discrete input space: **(E)** GFP (Sarkisyan et al., 2016b), where the goal is to generate the protein sequence with maximum fluorescence, **(F)** TF Bind 8, where the goal is to design a length 8 DNA sequence with high binding affinity with particular transcriptions factors and **(G)** UTR (Barrera et al., 2016), where the goal is to design a length 50 human 5'UTR DNA sequence with high ribosome loading. We represent discrete inputs in a transformed space of continuous-valued log probabilities for these tasks. Results for all baseline methods are based on numbers reported by Trabucco et al. (2021). Additional details for the setup of these tasks is provided in Appendix Section D.

Results on continuous tasks. Our results for different domains are shown in Table 1. On three out of four continuous tasks, COMs attain the best results, in some cases (e.g. (B) HopperController) attaining the performance of over **1.3x** the best prior method. In addition, COMs are shown to be the only method to attain higher performance that the best training point on every task. A naïve objective model without the conservative term, which is prone to falling off-the-manifold of valid inputs, struggles in especially high-dimensional tasks. Similarly, methods based on generative models, such as CbAS and MINs perform really poorly in the task of optimization over high-dimensional neural network weights in the HopperController task. 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 $y = \hat{f}_\theta(\mathbf{x})$ of the objective function with a regularizer, without any need for training a generative model, which may be harder in practice to effectively tune.

Results on tasks with a discrete input space. COMs perform competitively with the best performing methods on GFP and TF Bind8, clearly outperforming the best sample in the observed task dataset. COMs attain almost the best performance on the GFP task and outperform CbAS variants and MINs on the TF Bind8 task. In addition, COMs outperform prior methods on the UTR task, attaining performance within one standard deviation of the highest performing method on that task.

Overall, COMs attain the best performance on **4/7** tasks, achieving a normalized average objective value of **0.985**, improving over the next best method by **16%** on average.

6.2. Ablation Experiments

In this section, we perform an ablative experimental analysis of COMs to answer questions posed at the beginning of Section 6. First, we evaluate the efficacy of using conservative training for learning a model of the objective function by comparing COMs to a naïve gradient ascent baseline and show that COMs are more *stable*, i.e., the optimization performance of COMs is much less sensitive to the number of gradient ascent steps used for optimization. Second, we evaluate the effect of varying values of the Lagrange threshold τ in Equation 6. Third, we demonstrate the *consistency* of COMs by evaluating the sensitivity of the optimization performance with respect to the number of samples N, that are used to compute the evaluation metric $\max(\mathbf{x}_1^*, \cdots, \mathbf{x}_N^*)$.

COMs are more stable than naïve gradient ascent. In order to better compare COMs and a naïve objective model optimized using gradient ascent, we visualize the true objective value for each $\mathbf{x}_t$ encountered during optimization (t in Line 2, Algorithm 2) in Figure 3. Observe that a naïve objective model can attain good performance for a "hand-tuned" number of gradient ascent steps, but it soon degrades

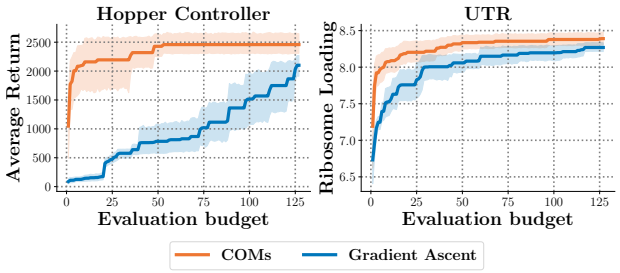

Figure 5. **Ablation of consistency of COMs by visualizing sensitivity to the post-optimization evaluation budget.** How does the performance of COMs and naïve gradient ascent vary as the evaluation budget is reduced? In our standard evaluation, we allow each offline MBO algorithm a "budget" of 128 evaluations for determining 100th and 50th percentile performance. The x-axis indicates the number N of allowed evaluations, and the y-axis indicates the 100th percentile performance of the chosen N points. As this evaluation budget is reduced, COMs is resilient, and remains superior to the naïve objective trained via supervised regression and optimized via standard gradient ascent. In the case of HopperController, COMs is nearly invariant to budgets down to size 55. This indicates COMs consistently produce optimized $\mathbf{x}^\star$ that attain high values under the true function.

in performance with more steps. This indicates that COMs are much more stable to the choice of number of gradient ascent steps performed than a naïve objective model.

Ablation of τ in Equation 6. In Figure 4, we evaluate the sensitivity of the performance of COMs as a function of the value of τ. As shown in Figure 4, we find that within the range of values evaluated, a higher value of τ gives rise to more stable optimization behavior, and we were able to utilize a universal value of $\tau = 0.5$ for all tasks with a continuous input space and $\tau = 2.0$ for all tasks with a discrete input space.

COMs consistently produce well-performing inputs. Finally, we evaluate the sensitivity of COMs to the evaluation procedure itself. Standard evaluation practice in offline MBO dictates evaluating a batch of N most promising candidate inputs produced by the algorithm with the ground truth objective, where N remains constant across all algorithms (Trabucco et al., 2021; Brookes et al., 2019), and using the maximum value attained over these inputs as the performance of the algorithm, i.e., $\max(\mathbf{x}_1^*, \cdots, \mathbf{x}_N^*)$. This measures if the algorithm performs well within a provided "evaluation budget" of N evaluations. An algorithm is more *consistent* if it attains higher values of the groundtruth function with a smaller value of the evaluation budget, N. We used $N = 128$ for evaluating all methods in Table 1, but the value of N is technically a hyperparameter and an effective offline MBO method should be resilient to this value, ideally. COMs are resilient to N: as we vary N from 1 to 128 in Figure 5, COMs not only perform well at larger values of N, but are also effective with smaller budgets, reaching near-optimal performance on HopperController in with a budget of 55, while a naïve objective model needs a budget twice as large to reach its own optimal performance, which is lower than that of COMs.

7. Discussion and Conclusion

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, we find that COMs give rise to good offline optimization performance and are considerably more stable than prior MBO methods, returning solutions that are comparable to and even better than the best existing MBO algorithms on four benchmark tasks. In this evaluation, COMs are consistently high performing, and in high-dimensional cases such as the Hopper Controller task, COMs improves on the next best method by a factor of **1.3x**. 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 close to the input manifold, and show improved stability at good solutions.

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. 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.

Acknowledgements

We thank anonymous ICML reviewers, Aurick Zhou and Justin Fu for discussions and feedback on the tasks and the method in this paper, and all other members from RAIL at UC Berkeley for their suggestions, feedback and support. This work was supported by National Science Foundation, the DARPA Assured Autonomy Program, C3.ai DTI, and compute support from Google, Microsoft and Intel.

References

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

Angermueller, C., Dohan, D., Belanger, D., Deshpande, R., Murphy, K., and Colwell, L. Model-based reinforcement learning for biological sequence design. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=HklxbgBKvr.

Barrera, L. A., Vedenko, A., Kurland, J. V., Rogers, J. M., Gisselbrecht, S. S., Rossin, E. J., Woodard, J., Mariani, L., Kock, K. H., Inukai, S., et al. Survey of variation in human transcription factors reveals prevalent dna binding changes. *Science*, 351(6280):1450–1454, 2016.

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

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

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

Fu, J. and Levine, S. Offline model-based optimization via normalized maximum likelihood estimation. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=FmMKSO4e8JK.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

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

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., and Teh, Y. W. Neural processes. *CoRR*, abs/1807.01622, 2018b. URL http://arxiv.org/abs/1807.01622.

Gaulton, A., Bellis, L. J., Bento, A. P., Chambers, J., Davies, M., Hersey, A., Light, Y., McGlinchey, S., Michalovich, D., Al-Lazikani, B., et al. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1):D1100–D1107, 2012.

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

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. NIPS'14, 2014a.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

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

Hamidieh, K. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, 154:346 – 354, 2018. ISSN 0927-0256. doi: https://doi.org/10.1016/j.commatsci.2018.07.052. URL http://www.sciencedirect.com/science/article/pii/S0927025618304877.

Hansen, N. The CMA evolution strategy: A comparing review. In Lozano, J. A., Larrañaga, P., Inza, I., and Bengoetxea, E. (eds.), *Towards a New Evolutionary Computation - Advances in the Estimation of Distribution Algorithms*, volume 192 of *Studies in Fuzziness and Soft Computing*, pp. 75–102. Springer, 2006. doi: 10.1007/3-540-32494-1_4. URL https://doi.org/10.1007/3-540-32494-1_4.

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

Hoburg, W. and Abbeel, P. Geometric programming for aircraft design optimization. volume 52, 04 2012. ISBN 978-1-60086-937-2. doi: 10.2514/6.2012-1680.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes. In *International Conference*

on Learning Representations, 2019. URL https://openreview.net/forum?id=SkE6PjC9KX.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes, 2013. URL http://arxiv.org/abs/1312.6114. cite arxiv:1312.6114.

Kumar, A. and Levine, S. Model inversion networks for model-based optimization. *NeurIPS*, 2019.

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

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

Liao, T., Wang, G., Yang, B., Lee, R., Pister, K., Levine, S., and Calandra, R. Data-efficient learning of morphology and controller for a microrobot. In *2019 IEEE International Conference on Robotics and Automation*, 2019. URL https://arxiv.org/abs/1905.01334.

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

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

Rao, R., Bhattacharya, N., Thomas, N., Duan, Y., Chen, P., Canny, J. F., Abbeel, P., and Song, Y. S. Evaluating protein transfer learning with TAPE. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 9686–9698, 2019. URL http://papers.nips.cc/paper/9163-evaluating-protein-transfer-learning-with-tape.

Rubinstein, R. Y. Optimization of computer simulation models with rare events. *European Journal of Operations Research*, 99:89–112, 1996.

Rubinstein, R. Y. and Kroese, D. P. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-carlo Simulation (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2004. ISBN 038721240X.

Sample, P. J., Wang, B., Reid, D. W., Presnyak, V., McFadyen, I. J., Morris, D. R., and Seelig, G. Human 5' utr design and variant effect prediction from a massively parallel translation assay. *Nature biotechnology*, 37(7): 803–809, 2019.

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

Sarkisyan, K. S., Bolotin, D. A., Meer, M. V., Usmanova, D. R., Mishin, A. S., Sharonov, G. V., Ivankov, D. N., Bozhanova, N. G., Baranov, M. S., Soylemez, O., Bogatyreva, N. S., Vlasov, P. K., Egorov, E. S., Logacheva, M. D., Kondrashov, A. S., Chudakov, D. M., Putintseva, E. V., Mamedov, I. Z., Tawfik, D. S., Lukyanov, K. A., and Kondrashov, F. A. Local fitness landscape of the green fluorescent protein. *Nature*, 533 (7603):397–401, May 2016a. ISSN 1476-4687. doi: 10.1038/nature17995. URL https://doi.org/10.1038/nature17995.

Sarkisyan, K. S., Bolotin, D. A., Meer, M. V., Usmanova, D. R., Mishin, A. S., Sharonov, G. V., Ivankov, D. N., Bozhanova, N. G., Baranov, M. S., Soylemez, O., Bogatyreva, N. S., Vlasov, P. K., Egorov, E. S., Logacheva, M. D., Kondrashov, A. S., Chudakov, D. M., Putintseva, E. V., Mamedov, I. Z., Tawfik, D. S., Lukyanov, K. A., and Kondrashov, F. A. Local fitness landscape of the green fluorescent protein. *Nature*, 533 (7603):397–401, May 2016b. ISSN 1476-4687. doi: 10.1038/nature17995. URL https://doi.org/10.1038/nature17995.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

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

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms.

In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, 2012. URL http://dl.acm.org/citation.cfm?id=2999325.2999464.

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

Thomas, P., Theocharous, G., and Ghavamzadeh, M. High-confidence off-policy evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Trabucco, B., Geng, X., Kumar, A., and Levine, S. Design-bench: Benchmarks for data-driven offline model-based optimization, 2021. URL https://github.com/brandontrabucco/design-bench.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I. J., Boneh, D., and McDaniel, P. D. 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. URL https://openreview.net/forum?id=rkZvSe-RZ.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, 1992a. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, May 1992b.

Wilson, J. T., Moriconi, R., Hutter, F., and Deisenroth, M. P. The reparameterization trick for acquisition functions. *CoRR*, abs/1712.00424, 2017. URL http://arxiv.org/abs/1712.00424.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. 2017. URL https://arxiv.org/abs/1611.01578.

Appendices

A. Method Details

In this section we provide additional information about our method **conservative objective models (COMs)**. In this section, we provide a 50th percentile evaluation of COMs compared to other methods and discuss additional details for COMs including including hyperparameters. Finally, we discuss how the benchmarking tasks are curated.

A.1. Additional Results

In addition to reporting performance using the mean 100th percentile objective value, as in Table 1, we additionally provide a table measuring the mean 50th percentile objective value in Table 2. This follows the convention for evaluation standardized by (Trabucco et al., 2021) when benchmarking model-based optimization algorithms. The 50th percentile results in Table 2 confirm that COMs again is optimal in **4/7** tasks, the most of any method we tested, and attains a normalized average performance of **0.590**, the greatest normalized average performance of all baselines we tested.

	GFP	TF Bind 8	UTR	Norm. avg. perf.	# Optimal
$\mathcal{D}$ (**best**)	0.789	0.439	0.593		
Auto. CbAS	0.848 ± 0.007	0.419 ± 0.007	0.576 ± 0.011	0.441	0 / 7
CbAS	0.852 ± 0.004	0.428 ± 0.010	0.572 ± 0.023	0.444	0 / 7
BO-qEI	0.246 ± 0.341	0.439 ± 0.000	0.571 ± 0.000	0.478	1 / 7
CMA-ES	0.047 ± 0.000	0.537 ± 0.014	**0.612 ± 0.014**	0.311	1 / 7
Grad.	0.838 ± 0.004	0.609 ± 0.019	0.593 ± 0.006	0.464	1 / 7
Grad. Min	0.837 ± 0.001	**0.645 ± 0.030**	0.598 ± 0.005	0.529	2 / 7
Grad. Mean	0.838 ± 0.002	**0.616 ± 0.023**	**0.601 ± 0.003**	0.528	3 / 7
MINs	0.820 ± 0.018	0.421 ± 0.015	0.585 ± 0.007	0.574	3 / 7
REINFORCE	0.844 ± 0.003	0.462 ± 0.021	0.568 ± 0.017	0.395	1 / 7
COMs (Ours)	**0.864 ± 0.000**	0.497 ± 0.038	**0.608 ± 0.012**	**0.590**	**4 / 7**

	Superconductor	Ant Morphology	D'Kitty Morphology	Hopper Controller	
$\mathcal{D}$ (**best**)	0.399	0.565	0.884	1.0	
Auto. CbAS	0.131 ± 0.010	0.364 ± 0.014	0.736 ± 0.025	0.019 ± 0.008	
CbAS	0.111 ± 0.017	0.384 ± 0.016	0.753 ± 0.008	0.015 ± 0.002	
BO-qEI	0.300 ± 0.015	0.567 ± 0.000	**0.883 ± 0.000**	0.343 ± 0.010	
CMA-ES	0.379 ± 0.003	-0.045 ± 0.004	0.684 ± 0.016	-0.033 ± 0.005	
Grad.	**0.476 ± 0.022**	0.134 ± 0.018	0.509 ± 0.200	0.092 ± 0.084	
Grad. Min	**0.471 ± 0.016**	0.185 ± 0.008	0.746 ± 0.034	0.222 ± 0.065	
Grad. Mean	**0.469 ± 0.022**	0.187 ± 0.009	0.748 ± 0.024	0.243 ± 0.064	
MINs	0.336 ± 0.016	**0.618 ± 0.040**	**0.887 ± 0.004**	**0.352 ± 0.058**	
REINFORCE	**0.463 ± 0.016**	0.138 ± 0.032	0.356 ± 0.131	-0.064 ± 0.003	
COMs (Ours)	0.386 ± 0.018	0.519 ± 0.026	**0.885 ± 0.003**	**0.375 ± 0.003**	

Table 2. **Comparative evaluation of COMs** against prior methods in terms of the mean 50th-percentile score and its standard deviation over 8 trials. Tasks include Superconductor, HopperController, AntMorphology, and DKittyMorphology, which have a continuous design input space and GFP, TFBind8 and UTR with a discrete design input space.

A.2. Implementation details

In addition to various considerations from Sections 3.3 and 3.4, one important implementation detail of COMs is to normalize the inputs ($\mathbf{x}$) and outputs (y values) for training the conservative model, $\hat{f}_\theta(\mathbf{x})$. Our motivation for using normalization was simple: Since the input and output ranges and modalities of various tasks we evaluated on in Table 1 is very different from each other, in order to be able to use a *uniform* set of hyperparameters for COMs, it is necessary to normalize both the

Hyperparameter	Discrete	Continuous
Number of epochs to train $\hat{f}_\theta$	50	50
T (Number of gradient ascent steps using Equation 5)	50	50
Number of steps used to generate adversarial samples $\mu(\mathbf{x})$ in Equation 6	50	50
α learning rate (used to optimize Equation 6 via dual gradient descent)	0.01	0.01
τ in Equation 6	2.0	0.5
η in Equation 5	$2.0\sqrt{d}$	$0.05\sqrt{d}$

Table 3. **Hyperparameters for COMs.** All hyperparameters are kept constant across all discrete tasks and continuous tasks respectively in COMs. The variable d indicates the cardinality of a single design $\mathbf{x}$ in the training set of the model. Scaling the learning rate by a factor proportional to $\sqrt{d}$ follows the implementation of the Gradient ascent baseline from Trabucco et al. (2021)

inputs $\mathbf{x}$ and outputs y to a standard range. Following standard normalization practices, we normalized $\mathbf{x}$ and y such that the resulting first and second moments match those of a unit Gaussian distribution. In practice, this means collecting all objective values from the training dataset into a vector $Y \in \mathbb{R}^{N \times 1}$, evaluating the sample mean $\hat{\mu} = \text{mean}(Y)$ and sample standard deviation $\hat{\sigma} = \text{std}(Y - \hat{\mu})$. A similar procedure is used for calculating the sample mean and sample standard deviation of $\mathbf{x}$. The objective values and inputs are then normalized by subtracting their sample mean and dividing by their sample standard deviation $y \leftarrow (y - \hat{\mu})/\hat{\sigma}$, except where doing so would divide by zero. This normalization allows COMs to use the uniform set of hyperparameters, which we mention explicitly, in Table 3.

A.3. Benchmarking Details

In order to promote reproducibility, we additionally provide the task identifiers and keyword arguments used with the *design-bench* Trabucco et al. (2021) package. These arguments are passed to the `design_bench.make` function call in order to build a model-based optimization Task object in Python. Note that in addition to specifying the name of the task dataset (such as GFP), one must also specify the desired oracle function (such as a Transformer). In Table 4 we detail the specific combination of task datasets and oracle functions used in this work. Additionally, when an approximate oracle is used, commonly because an exact simulator or closed form equation for the ground truth y values is not available, there is a train-test discrepancy, where the predictions of the approximate oracle may not be a perfect reflection of the ground truth y values contained in the original model-based optimization dataset. This discrepancy is further explored by Trabucco et al. (2021); however, we find that UTR is particular susceptible to such discrepency, and so we choose to relabel the y values contained in the MBO dataset with the predictions of the CNN oracle. See Appendix D for more information.

Task	Design-Bench ID	Relabel
GFP	`GFP-Transformer-v0`	False
TF Bind 8	`TFBind8-Exact-v0`	False
UTR	`UTR-ResNet-v0`	True
Superconductor	`Superconductor-RandomForest-v0`	False
Ant Morphology	`AntMorphology-Exact-v0`	False
D'Kitty Morphology	`DKittyMorphology-Exact-v0`	False
Hopper Controller	`HopperController-Exact-v0`	False

Table 4. **Design-Bench task identifiers.** This table contains the necessary arguments to pass to the `design_bench.make` function call. More information is available at `https://github.com/brandontrabucco/design-baselines`, and documentation for Design-Bench is available at `https://github.com/brandontrabucco/design-bench`

B. Proof of Theorem 1

In this section, we provide a proof for Theorem 1 and show that the conservative training by performing gradient descent on θ with respect to the objective in Equation 7 (restated below in a more convenient form as Equation 8) indeed obtains a conservative model of the actual objective function. Note that $\overline{\mathcal{D}}(\mathbf{x}'|\mathbf{x})$ denotes a smoothed Dirac-delta distribution centered

at $\mathbf{x}$, which can be obtained by adding random noise to a given $\mathbf{x}$.

$$\mathcal{L}(\theta; \mu, \overline{\mathcal{D}}) := \alpha \left(\mathbb{E}_{\mathbf{x}_0 \sim \overline{\mathcal{D}}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)} \left[\hat{f}_\theta(\mathbf{x}_T) \right] - \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}, \mathbf{x}_T \sim \overline{\mathcal{D}}(\mathbf{x}_T|\mathbf{x}_0)} \left[\hat{f}_\theta(\mathbf{x}_T) \right] \right) + \underbrace{\frac{1}{2} \mathbb{E}_{\mathbf{x}_0 \sim \overline{\mathcal{D}}, (\mathbf{x}, y) \sim \overline{\mathcal{D}}(\mathbf{x}|\mathbf{x}_0)} \left[\left(\hat{f}_\theta(\mathbf{x}) - y \right)^2 \right]}_{:= (\wedge)}. \tag{8}$$

We now restate a formal version of Theorem 1 (including correcting a typo from the informal version in the main paper, where the argument of the left hand side of the expression was denoted as $\mathbf{x}'$ instead of $\mathbf{x}''$), and then provide a proof. We make an additional assumption that the neural tangent kernel, $\mathbf{G}_f^k(\mathbf{x}, \mathbf{x}')$, is semi-positive definite.

Theorem 2 (Formal version of Theorem 1)**.** *Assume that $\hat{f}_\theta(\mathbf{x})$ is trained by performing gradient descent on θ with respect to the objective $\mathcal{L}(\theta; \mu, \overline{\mathcal{D}})$ in Equation 8 with a learning rate η. The parameters in step k of gradient descent are denoted by θ^k, and let the corresponding conservative model be denoted as $\hat{f}_\theta^k$. Let $\mathbf{G}$, μ, $\widehat{L}$, L, $\overline{\mathcal{D}}$ be defined as discussed above. Then, under assumptions listed above, $\forall\, \mathbf{x} \in \mathcal{D}, \mathbf{x}'' \in \mathcal{X}$, the conservative model at iteration $k+1$ of training satisfies:*

$$\hat{f}_\theta^{k+1}(\mathbf{x}'') := \max \left\{ \hat{f}_\theta^{k+1}(\mathbf{x}) - \widehat{L} ||\mathbf{x}'' - \mathbf{x}||_2, \tilde{f}_\theta^{k+1}(\mathbf{x}'') - \eta\alpha \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}, \mathbf{x}' \sim \mu}[\mathbf{G}_f^k(\mathbf{x}'', \mathbf{x}')] + \eta\alpha \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}, \mathbf{x}' \sim \overline{\mathcal{D}}}[\mathbf{G}_f^k(\mathbf{x}'', \mathbf{x}')] \right\},$$

where $\tilde{f}_\theta^{k+1}(\mathbf{x}'')$ is the resulting $(k+1)$-th iterate of $\hat{f}_\theta$ if conservative training were not used. Thus, if α is sufficiently large, the expected value of the asymptotic function, $\hat{f}_\theta := \lim_{k \to \infty} \hat{f}_\theta^k$, on inputs $\mathbf{x}_T$ found by the optimizer, lower-bounds the value of the true function $f(\mathbf{x}_T)$:

$$\mathbb{E}_{\mathbf{x}_0 \sim \mathcal{D}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[\hat{f}_\theta(\mathbf{x}_T)] \leq \mathbb{E}_{\mathbf{x}_0 \sim \mathcal{D}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[f(\mathbf{x})].$$

Proof. For proving the first part of the theorem, we first derive the expression for the gradient of $\mathcal{L}(\theta; \mu, \overline{\mathcal{D}})$ with respect to θ, and denote the y-value for a given $\mathbf{x}$ as a deterministic function $y(\mathbf{x})$. Our proof can directly be extended to a non-deterministic $y(\mathbf{x})$ with an additional integral over y values, but we stick to deterministic $y(\mathbf{x})$ for simplicity.

$$\nabla_\theta \mathcal{L}(\theta; \mu, \overline{\mathcal{D}}) = \alpha \int \left(\overline{\mathcal{D}}(\mathbf{x}_0) \mu(\mathbf{x}|\mathbf{x}_0) - \overline{\mathcal{D}}(\mathbf{x}_0) \overline{\mathcal{D}}(\mathbf{x}|\mathbf{x}_0) \right) \nabla_\theta \hat{f}_\theta(\mathbf{x})\, d\mathbf{x}_0 d\mathbf{x} + \int \overline{\mathcal{D}}(\mathbf{x}_0) \overline{\mathcal{D}}(\mathbf{x}|\mathbf{x}_0) (f_\theta(\mathbf{x}) - y(\mathbf{x})) \nabla_\theta \hat{f}_\theta(\mathbf{x})\, d\mathbf{x} d\mathbf{x}_0.$$

At any iteration k of gradient descent, the next parameter iterate θ^{k+1} are obtained via, $\theta^{k+1} = \theta^k - \eta \nabla_\theta \mathcal{L}(\theta; \mu, \overline{\mathcal{D}})$. Using this relation, and making an approximate linearization assumption on the non-linear function $\hat{f}_\theta^k$ for a small learning rate $\eta << 1$ under the assumption of the neural tangent kernel (NTK) (Jacot et al., 2018) regime, which models the behavior of deep neural networks in the infinite-width limit, we obtain the expression for the next function value: $\hat{f}_\theta^{k+1}(\mathbf{x}'')$:

$$\begin{aligned} &\hat{f}_\theta^{k+1}(\mathbf{x}'') \approx \hat{f}_\theta^k(\mathbf{x}'') + (\theta^{k+1} - \theta^k)^T \nabla_\theta \hat{f}_\theta^k(\mathbf{x}'') \\ &= \underbrace{\hat{f}_\theta^k(\mathbf{x}'') + \eta \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}, \mathbf{x}' \sim \overline{\mathcal{D}}}[\left(y(\mathbf{x}') - \hat{f}_\theta^k(\mathbf{x}') \right) \mathbf{G}_f^k(\mathbf{x}'', \mathbf{x}')]}_{:= (*)} - \underbrace{\left(\eta\alpha \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}, \mathbf{x}' \sim \mu}[\mathbf{G}_f^k(\mathbf{x}'', \mathbf{x}')] - \eta\alpha \mathbb{E}_{\mathbf{x} \sim \overline{\mathcal{D}}, \mathbf{x}' \sim \overline{\mathcal{D}}}[\mathbf{G}_f^k(\mathbf{x}'', \mathbf{x}')] \right)}_{:= \Delta(\mathbf{x}'')}, \end{aligned}$$

where the expression marked as $(*)$ denotes the $(k+1)$-th iterate of the function, under gradient descent on just the mean-squared error $(f(\mathbf{x}) - y)^2$ term, marked as $(\wedge)$ in Equation 8. Noting that the theorem statement denotes the term $(*)$ as $\tilde{f}_\theta^{k+1}(\mathbf{x}'')$, we obtain our first desired result. To obtain the first argument of the $\max$ in the theorem statement, note that if the function $\hat{f}_\theta^{k+1}$ is $\widehat{L}$-Lipschitz, the value at $\mathbf{x}''$ cannot be smaller than $\hat{f}_\theta^{k+1}(\mathbf{x}) - \widehat{L} ||\mathbf{x} - \mathbf{x}''||_2$, and hence the maximum over the two terms.

For proving the second part of the theorem statement, observe that if we can show that in expectation over $\mathbf{x}'' \sim \mu(\mathbf{x}_T); \mu(\mathbf{x}_T) := \int_{\mathbf{x}_0} \overline{\mathcal{D}}(\mathbf{x}_0) \mu(\mathbf{x}_T|\mathbf{x}_0)\, d\mathbf{x}_0$, the quantity $\Delta(\mathbf{x}'')$ is positive, then our argument is complete since we have shown that each step of gradient descent on θ reduces the value of $\mathbb{E}_{\mathbf{x}_0 \sim \overline{\mathcal{D}}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[\hat{f}_\theta^k(\mathbf{x}_T)]$ by a positive quantity by virtue of training with Equation 8 as compared to only training θ with standard squared error $(\wedge)$. Thus, if $\mathbb{E}_{\mathbf{x}_0 \sim \overline{\mathcal{D}}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[\Delta^k(\mathbf{x}_T)]$ is positive for all gradient descent steps k, we obtain the desired lower-bound condition as $k \to \infty$. As an additional detail, note that we assumed $\widehat{L} >> L$ (i.e. the Lipschitz constant of $\hat{f}_\theta(\mathbf{x})$ is sufficiently larger than that of $f(\mathbf{x})$). This condition handles the boundary case when the predictions $\hat{f}_\theta^{k+1}(\mathbf{x}')$ get lower-bounded under the first argument of $\max$ in the first part of Theorem 2 due to the Lipschitz condition: $\hat{f}_\theta^{k+1}(\mathbf{x}) - \widehat{L} ||\mathbf{x}' - \mathbf{x}||_2$.

Finally, we fill in the missing piece that show $\mathbb{E}_{\mathbf{x}_0 \sim \overline{\mathcal{D}}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[\Delta^k(\mathbf{x}_T)]$ is positive for each k. Under the assumption that the neural tangent kernel $\mathbf{G}^k(\mathbf{x}, \mathbf{x}')$ is semi-positive definite for all k, we can express:

$$\begin{aligned}\mathbb{E}_{\mathbf{x}_0 \sim \overline{\mathcal{D}}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[\Delta^k(\mathbf{x}_T)] &:= \eta\alpha \int_{\mathbf{x}, \mathbf{x}_0, \mathbf{x}', \mathbf{x}_T} \left[\overline{\mathcal{D}}(\mathbf{x})\mu(\mathbf{x}'|\mathbf{x}) - \overline{\mathcal{D}}(\mathbf{x})\overline{\mathcal{D}}(\mathbf{x}'|\mathbf{x})\right] \overline{\mathcal{D}}(\mathbf{x}_0)\mu(\mathbf{x}_T|\mathbf{x}_0)\mathbf{G}_f^k(\mathbf{x}', \mathbf{x}_T) \\ &= \eta\alpha \int_{\mathbf{x}_0} \overline{\mathcal{D}}(\mathbf{x}_0) \int_{\mathbf{x}} \overline{\mathcal{D}}(\mathbf{x}) \int_{\mathbf{x}', \mathbf{x}_T} [\mu(\mathbf{x}'|\mathbf{x}) - \overline{\mathcal{D}}(\mathbf{x}'|\mathbf{x})]\mu(\mathbf{x}_T|\mathbf{x}_0)\mathbf{G}_f^k(\mathbf{x}', \mathbf{x}_T)\end{aligned}$$

By now writing the above in matrix form, we note that the RHS of the above equation has the same structure as the second term in the RHS of Equation 14 in Kumar et al. (2020), and furthermore since $\mathbf{G}_f^k$ is positive semi-definite, it satisfies the required conditions for Equation 14 and Theorem D.1 from Kumar et al. (2020) to be applicable. Thus, exactly following the proof of Theorem D.1 in Kumar et al. (2020) for the linear function approximation case in reinforcement learning, with the following substitutions: $P_{\mathbf{F}} := \mathbf{G}_f^k(\cdot, \mathbf{x}_T)$ (i.e., a column of the kernel Gram-matrix for a fixed value of the second argument) and $a = \mathbf{x}_T$, $s = \mathbf{x}_0$, we can show that $\mathbb{E}_{\mathbf{x}_0 \sim \overline{\mathcal{D}}, \mathbf{x}_T \sim \mu(\mathbf{x}_T|\mathbf{x}_0)}[\Delta^k(\mathbf{x}_T)] \geq 0$, thus finishing our argument. □

C. Network Details

In each of our experiments, we train a neural network $\hat{f}_\theta$ to approximate the ground truth score function of an offline MBO task, where θ represents the weights of the model. Distinct from prior methods based on generative models (Kumar & Levine, 2019; Brookes et al., 2019) we are able to utilize the same neural network architecture for representing the learned model, $\hat{f}_\theta(\mathbf{x})$ across all MBO tasks. This architecture is a three-layer neural network with two hidden layers of size 2048, followed by Leaky ReLU activation functions with a leak of 0.3. Each neural network $\hat{f}_\theta$ has an output layer that predicts a single scalar objective value y, which is used for regression. Specifically, $\hat{f}_\theta$ is trained to minimize the mean squared error of observed objective values, using the default parameters of the Adam optimizer as discussed in Section 3.4.

D. Data Collection

In this section, we detail the data collection steps used for creating each of the tasks from (Trabucco et al., 2021), used for benchmarking COMs. We answer **(1)** where is the data from, and **(2)** what pre-processing steps are used?

D.1. TF Bind 8

The TF Bind 8 task is a derivative of the transcription factor binding activity survey performed by Barrera et al. (2016), where the binding activity scores of every possible length eight DNA sequence was measured with a variety of human transcription factors. We filter the dataset by selecting a particular transcription factor `SIX6_REF_R1`, and defining an optimization problem where the goal is to synthesize a length 8 DNA sequence with high binding activity with human transcription factor `SIX6_REF_R1`. This particular transcription factor for TF Bind 8 was recently used for optimization in Angermueller et al. (2020b;a). TF Bind 8 is a fully characterized dataset containing 65792 samples, representing every possible length 8 combination of nucleotides $\mathbf{x}_{\text{TFBind8}} \in \{0, 1\}^{8 \times 4}$. The training set given to offline MBO algorithms is restricted to the bottom 50%, which results in a visible training set of 32898 samples.

D.2. GFP

The GFP task provided is a derivative of the GFP dataset (Sarkisyan et al., 2016b). The dataset we use in practice is that provided by Brookes et al. (2019) 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 $\mathbf{x}_{\text{GFP}} \in \{0, 1\}^{237 \times 20}$. This tensor is a sequence of 237 one-hot vectors corresponding to which amino acid is present in that location in the protein. We use the dataset format of (Brookes et al., 2019) 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. We employ the full dataset of 56086 proteins when learning approximate oracles for evaluating offline MBO methods, but restrict the training set given to offline MBO algorithms to 5000 samples drawn from between the 50th percentile and 60th percentile of proteins in the GFP dataset, sorted by fluorescence values. This subsampling procedure is consistent with the procedure used by prior work (Brookes et al., 2019).

D.3. UTR

The UTR task is derived from work by Sample et al. (2019) who trained a CNN model to predict the expressive level of a particular gene from a corresponding 5'UTR sequence. Our use of the UTR task for model-based optimization follows Angermueller et al. (2020a), where the goal is to design a length 50 DNA sequence to maximize expression level. We follow the methodology set by Sample et al. (2019) to sort all length 50 DNA sequences in the unprocessed UTR dataset by total reads, and then select the top 280,000 DNA sequences with the most total reads. The result is a dataset containing 280,000 samples of length 50 DNA sequences $\mathbf{x}_{\text{UTR}} \in \{0, 1\}^{50\times 4}$ and corresponding ribosome loads. When training offline MBO algorithms, we subsequently eliminate the top 50% of sequences ranked by their ribosome load, resulting in a visible dataset with only 140,000 samples.

D.4. Superconductor

The Superconductor task is inspired by recent work (Fannjiang & Listgarten, 2020) 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 (Hamidieh, 2018), and subsequently made available to the public at `https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data#`. The original dataset consists of superconductors featurized into vectors containing measured physically properties like the number of chemical elements present, or the mean atomic mass of such elements. One issue with the original dataset that was used in (Fannjiang & Listgarten, 2020) is that the numerical representation of the superconducting materials did not lend itself to recovering a physically realizable material that could be synthesized in a lab after performing model-based optimization. In order to create an *invertible* input specification, we deviate from prior work and encode superconductors as vectors whose components represent the number of atoms of specific chemical elements present in the superconducting material—a serialization of the chemical formula of each superconductor. The result is a real-valued design space with 86 components $\mathbf{x}_{\text{Superconductor}} \in \mathcal{R}^{86}$. The full dataset used to learn approximate oracles for evaluating MBO methods has 21263 samples, but we restrict this number to 17010 (the 80th percentile) for the training set of offline MBO methods to increase difficulty.

D.5. Hopper Controller

The goal of the Hopper Controller task is to design a set of weights for a neural network policy that achieves high expected return. 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 (Schulman et al., 2017). Specifically, we use the default parameters for PPO provided in stable baselines (Hill et al., 2018). 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 $\mathbf{x}_{\text{Hopper}} \in \mathcal{R}^{5126}$. 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, which is standard practice for this MuJoCo environment.

D.6. Ant & D'Kitty Morphology

Both morphology tasks 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 of a pre-trained morphology conditioned agent. We implement this by pre-training a morphology conditioned neural network policy using SAC (Haarnoja et al., 2018). For both the Ant and the D'Kitty, we train the agents for more than ten million environment steps, and a maximum episode length of 200, with all other settings as default. These agents are pre-trained on Gaussian distributions of morphologies. The Gaussian distributions are obtained by adding Gaussian noise with standard deviation 0.03 for Ant and 0.01 for D'Kitty the design-space range to the default morphologies.

After obtaining trained morphology-conditioned policies, we create a dataset of morphologies for model-based optimization by sampling initialization points randomly, and then using CMA-ES to optimize for morphologies that attain high reward using the pretrained morphology-conditioned policy. To obtain initialization points, we add Gaussian random noise to the

default morphology for the Ant with standard deviation 0.075 and D'Kitty with standard deviation 0.1, and then apply CMA-ES with standard deviation 0.02. We ran CMA-ES for 250 iterations and then restart, until a minimum of 25000 morphologies were collected, resulting in a final dataset size of 25009 for both the Ant and D'Kitty. The design space for Ant Morphologies is $\mathbf{x}_{\text{Ant}} \in \mathcal{R}^{60}$, whereas for D'Kitty morphologies is $\mathbf{x}_{\text{D'Kitty}} \in \mathcal{R}^{56}$. We subsample the dataset to its 40th percentile when training offline MBO algorithms, resulting in 10004 samples.

E. Oracle Functions

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

E.1. TF Bind 8

TF Bind 8 is a fully characterized discrete offline MBO task, which means that all possible designs have been evaluated (Barrera et al., 2016) and are contained in the full hidden TF Bind 8 dataset. The oracle for TF Bind 8 is therefore implemented as a lookup table that returns the score corresponding to a particular length 8 DNA sequence from the dataset. By restricting the size of the training set visible to an offline MBO algorithm, it is possible for the algorithm to propose a design that achieves a higher score than any other DNA sequence visible to the offline MBO algorithm during training.

E.2. GFP

GFP uses a simplified Transformer to the TAPE Transformer proposed by Rao et al. (2019). The Transformer used has 4 attention blocks and a hidden size of 64. The Transformer is fit to the entire hidden GFP dataset, making it possible to sample a protein design that achieves a higher score than any other protein visible to an offline MBO algorithm. The model has a Spearman's rank correlation coefficient of 0.8497 with a held-out validation set derived from the GFP dataset.

E.3. UTR

UTR uses a CNN, which differs from the CNN that was originally used by (Angermueller et al., 2020a) in that it has residual connections. Our reasoning for making this change is that ResNet is a newer and possibly higher capacity model that may be less prone to mistakes than the shallower CNN model proposed by Sample et al. (2019). The chosen CNN has 2 residual blocks with 2 convolution layer each, and a hidden size of 120. The CNN is fit to the entire hidden UTR dataset, making it possible to sample a DNA sequence that achieves a higher score than any other sequence visible to an offline MBO algorithm. The resulting CNN has a spearman's rank correlation coefficient of 0.8617 with a held-out validation set.

E.4. Superconductor

The Superconductor oracle function is also a random forest regression model. The model we use it the model described by (Hamidieh, 2018). We borrow the hyperparameters described by them, and we use the RandomForestRegressor provided in scikit-learn. Similar to the setup for the previous set of tasks, this oracle is trained on the entire hidden dataset of superconductors. The random forest has a spearman's rank correlation coefficient with a held-out validation set of 0.9155.

E.5. HopperController

HopperController and the remaining tasks implement an exact oracle function. For HopperController the oracle takes the form of a single rollout using the Hopper-v2 MuJoCo environment. The designs for HopperController 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 Hopper Controller oracle. The limit of performance is the maximum return that an agent can achieve in Hopper-v2 over 1000 steps.

E.6. Ant & D'Kitty Morphology

The final two tasks in design-bench use an exact oracle function, using the MuJoCo simulator. For both morphology tasks, the simulator performs a rollout and returns the sum of rewards at every timestep in that rollout. Each task is accompanied by

a pre-trained morphology-conditioned 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 simulations can be time consuming to run, and so we limit the rollout length to 100 steps. The morphology conditioned policies were trained using Soft Actor Critic for 10 million steps for each task, and are ReLU networks with two hidden layers of size 64.