Design-Bench: Benchmarks for Data-Driven Offline Model-Based Optimization

Brandon Trabucco [* 1]   Xinyang Geng [* 2]   Aviral Kumar [2]   Sergey Levine [2]

Abstract

Black-box model-based optimization (MBO) problems, where the goal is to find a design input that maximizes an unknown objective function, are ubiquitous in a wide range of domains, such as the design of proteins, DNA sequences, aircraft, and robots. Solving model-based optimization problems typically requires actively querying the unknown objective function on design proposals, which means physically building the candidate molecule, aircraft, or robot, testing it, and storing the result. This process can be expensive and time consuming, and one might instead prefer to optimize for the best design using only the data one already has. This setting—called offline MBO—poses substantial and different algorithmic challenges than more commonly studied online techniques. A number of recent works have demonstrated success with offline MBO for high-dimensional optimization problems using high-capacity deep neural networks. However, the lack of standardized benchmarks in this emerging field is making progress difficult to track. To address this, we present Design-Bench, a benchmark for offline MBO with a unified evaluation protocol and reference implementations of recent methods. Our benchmark includes a suite of diverse and realistic tasks derived from real-world optimization problems in biology, materials science, and robotics that present distinct challenges for offline MBO. Our benchmark and reference implementations are released at github.com/rail-berkeley/design-bench and github.com/rail-berkeley/design-baselines.

*Equal contribution [1]Machine Learning Department, Carnegie Mellon University [2]Department of Electrical Engineering and Computer Sciences, University of California Berkeley. Correspondence to: Brandon Trabucco <btrabucco@berkeley.edu>, Xinyang Geng <young.geng@berkeley.edu>.

1. Introduction

Automatically synthesizing designs that maximize a desired objective function is one of the most important challenges in scientific and engineering disciplines. From protein design in molecular biology (Shen et al., 2014) to superconducting material discovery in physics (Hamidieh, 2018), researchers have made significant progress in applying machine learning to optimization problems over structured design spaces.

Commonly, the exact form of the objective function is unknown, and the objective value for a novel design can only be found by either running computer simulations or real world experiments. This process of optimizing an unknown function by only observing samples from this function is known as black-box optimization, and is typically solved in an **online** iterative manner, where in each iteration the solver proposes new designs and queries the objective function for feedback in order to inform better design proposals at the next iteration (Williams & Rasmussen, 2006). In many domains however, the objective function is prohibitively expensive to evaluate because it requires manually conducting experiments in the real world. In this setting, one cannot query the true objective function, and cannot receive feedback on design proposals. Instead, a collection of past records of designs and corresponding objective values might be available, and the optimization method must instead leverage existing data to synthesize the most optimal designs. This is called **offline model-based optimization** (offline MBO).
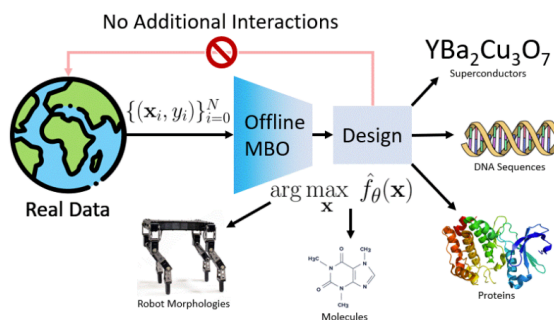

Figure 1. **Offline model-based optimization** (MBO) requires generating designs $\mathbf{x}$ that optimize a black-box objective function $f(\mathbf{x})$ using a given static dataset of designs, without any active queries to the ground truth function.

Although online black-box optimization has been studied extensively, offline MBO has received comparatively less

attention, and only a small number of recent works study offline MBO in the setting with high-dimensional design spaces (Brookes et al., 2019; Kumar & Levine, 2019; Fannjiang & Listgarten, 2020; Fu & Levine, 2021; Trabucco et al., 2021). This is partly because online techniques cannot be directly applied in settings where offline MBO is used, especially in high-dimensional settings. Online techniques, such as Bayesian optimization (Snoek et al., 2012), often require iterative feedback via queries to the objective function. Such online optimizers exhibit optimistic behavior: they rely on active queries at completely unseen designs irrespective of whether such a design is good or not. When access to these queries is removed, certain considerations change: optimism is no longer desirable and distribution shift becomes a major challenge (Kumar & Levine, 2019).

Even with only a few existing offline MBO methods, it is hard to compare and track progress, as methods are typically proposed and evaluated on different tasks with distinct evaluation protocols. To the best of our knowledge, there is no commonly adopted benchmark for offline MBO. To address, we introduce a suite of tasks for offline MBO with a standardized evaluation protocol. We include a diverse set of tasks that span a wide range of application domains—from synthetic biology to robotics–that aims at representing the core challenges in real-world offline MBO. While the tasks are not intended to directly enable solving the corresponding real-world problems, which would require a lot of machinery in real hardware setup (e.g., a real robot or access to a wetlab for molecule design), they are intended to provide algorithm designers with a representative sampling of challenges that reflect the difficulties with real-world MBO. That is to say, the tasks are not intended to be *real*, but are intended to be *realistically challenging*. Further, the diversity of the tasks measures how they generalize across multiple domains and verifies they are not specialized to a single task. Our benchmark incorporates a variety of challenging factors, such as high dimensionality and sensitive discontinuous objective functions, which help identify the strengths and weaknesses of MBO methods. Along with this benchmark suite, we present reference implementations of a number of existing offline MBO and baseline optimization methods. We systematically evaluate them on all of the proposed benchmark tasks and report results. We hope that our work can provide insight into the progress of offline MBO methods and serve as a meaningful metric to galvanize research in this area.

2. Offline Model-Based Optimization (Offline MBO) Problem Statement

In online model-based optimization, the goal is to optimize a (possibly stochastic) black-box objective function $f(\mathbf{x})$ with respect to its input. The objective can be written as

$\arg\max_{\mathbf{x}} f(\mathbf{x})$. Methods for online MBO typically optimize the objective iteratively, proposing design $\mathbf{x}_k$ at the kth iteration and query the objective function to obtain $f(\mathbf{x}_k)$. Unlike its online counterpart, access to the true objective f is not available in offline MBO. Instead, the algorithm $\mathfrak{A}$ is provided access to a static dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ of designs $\mathbf{x}_i$ and a corresponding measurement of the objective value y_i. The algorithm consumes this dataset and produces an optimized candidate design $\mathbf{x}^*$ which is evaluated against the true objective function. This paradigm is illustrated in Figure 1. Abstractly, the objective for offline MBO is:

$$\arg\max_{\mathfrak{A}} f(\mathbf{x}^*) \text{ where } \mathbf{x}^* = \mathfrak{A}(\mathcal{D}). \qquad (1)$$

In practice, producing a single optimal design entirely from offline data is very difficult, so offline MBO methods are more commonly evaluated (Kumar & Levine, 2019) in terms of "P percentile of top K" performance, where the algorithm produces K candidates and the P percentile objective value determines final performance. Next we discuss two important aspects pertaining to offline MBO, namely, why offline MBO algorithms can improve beyond the best design observed in the offline dataset despite no active queries, and the associated challenges with devising offline model-based optimization algorithms.
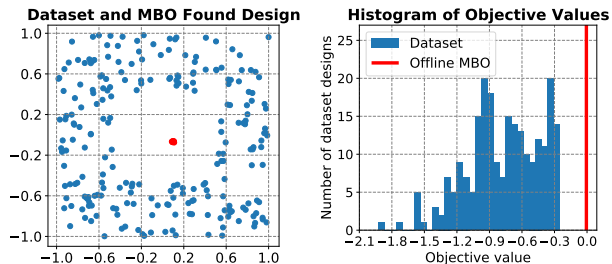

Figure 2. Offline MBO finds designs better than the best in the observed dataset by exploiting compositional structure of the objective function. **Left:** datapoints in a toy quadratic function MBO task over 2D space with optimum at $(0.0, 0.0)$ in blue, MBO found design in red. **Right:** Objective value for optimal design is much higher than that observed in the dataset.

Would offline MBO even produce designs better than the best observed design in the dataset?
A natural question to ask is whether it is even reasonable to expect offline MBO algorithms to improve over the performance of the best design seen in the dataset. As we will show in our benchmark results, many of the tasks that we propose do already admit solutions from existing algorithms that exceed the performance of the best sample in the dataset. To provide some intuition for how this can be possible, consider a simple example of offline MBO problems, where the objective function $f(\mathbf{x})$ can be represented as a sum of functions of independent partitions of the design variables, i.e., $f(\mathbf{x}) = f_1(\mathbf{x}[1]) + f_2(\mathbf{x}[2]) + \cdots + f_N(\mathbf{x}[N]))$, where $\mathbf{x}[1], \cdots, \mathbf{x}[N]$ denotes disjoint subsets of design

variables $\mathbf{x}$. The dataset of the offline MBO problem contains optimal design variable for each partition, but not the combination. If an offline MBO algorithm can identify the compositional structure of independent partitions, it would be able to combine the optimal design variable for each partition together to form the overall optimal design and therefore improving the performance over the best design in the dataset. To better demonstrate this idea, we created a toy problem in two dimensions, where the objective function is simply $f(x, y) = -x^2 - y^2$. We then run a naïve gradient ascent algorithm, as we will describe later in this paper. In Figure 2, we can clearly see that our offline MBO algorithm is able to learn to combine the best x and y and produce designs significantly better than the best sample in the dataset. Such a condition appears in a number of scenarios in practice e.g., in reinforcement learning (RL), where the Markov structure provides a natural decomposition satisfying this composition criterion (Fu et al., 2020) and effective offline RL algorithms are known to exploit this structure (Fu et al., 2020) or in protein design, where objective such as fluorescence naturally decompose into functions of neighboring amino acids (Brookes et al., 2019).

What makes offline MBO especially challenging?

The offline nature of the problem prevents the algorithm $\mathfrak{A}$ from querying the ground truth objective f with its proposed design candidates, and this makes the offline MBO problem much more difficult than the online design optimization problem. One naïve approach to tackle this problem is to learn a model of the objective function using the dataset, which we can denote $\hat{f}(\mathbf{x})$, and then convert this offline MBO problem into a regular online MBO problem by treating the learned objective model as the true objective. However, this generally does not work: optimizing the design $\mathbf{x}$ with respect to a learned proxy $\hat{f}(\mathbf{x})$ will produce *out-of-distribution* designs that "fool" $\hat{f}(\mathbf{x})$ into outputting a high value, analogously to adversarial examples. Indeed, it is well known that optimizing naïvely with respect to model inputs to obtain a desired output will usually simply "fool" the model (Kumar & Levine, 2019). A naïve strategy to address this out-of-distribution issue is to constrain the design to stay close to the data, but this is also problematic, since in order to produce a design that is better than the best training point, it is usually necessary to deviate from the training data at least somewhat. In almost all practical MBO problems, such as optimization over drug molecules or robot morphologies as we discuss in section 5, designs with the highest objective values typically lie on the tail of the dataset distribution and we may not find them by staying extremely close to the data distribution. This conflict between the need to remain close to the data to avoid out-of-distribution inputs and the need to deviate from the data to produce better designs is one of the core challenges of offline MBO. This challenge is often exacerbated in real-world settings by the

high dimensionality of the design space and the sparsity of the available data, as we will show in our benchmark. A good offline MBO method needs to carefully balance these two sides, producing optimized designs that are good, but not too far from the data distribution.

3. Related Work

Prior work has extensively focused on online or active MBO, which requries active querying on the ground truth function, including algorithms using Bayesian optimization and their scalable variants (Lizotte, 2008; Snoek et al., 2012; 2015; Shahriari et al., 2015; Perrone et al., 2017), direct search (Kolda et al., 2003), genetic or evolutionary algorithms (Whitley, 1994; Pal et al., 2012; Yang & Slowik, 2020), the cross-entropy method (Rubinstein & Kroese, 2013), simulated annealing (Van Laarhoven & Aarts, 1987), etc. These methods may not be well suited for real-world problems where the ground truth function is expensive to evaluate and therefore prohibitive for active querying. Offline MBO utilizes an already existing database of designs and objective values, which might be obtained from previously conducted experiments. This presents an attractive algorithmic paradigm towards approaching such scenarios. Since offline MBO prohibits any ability to query the true objective with new designs, it presents different challenges from those typically studied in online MBO problem, as we discuss in Section 5. These new challenges in turn require new benchmarks, motivating our work.

The most important components for a good offline MBO benchmark are datasets that capture the challenges of real-world problems. Fortunately, researchers working on a wide variety of scientific fields have already collected many datasets of designs which we can use for training offline MBO algorithms. ChEMBL (Gaulton et al., 2012) provides a dataset for drug discovery, where molecule activities are measured against a target assay. Hamidieh (2018) analyze the critical temperatures for superconductors and provide a dataset to search for room-temperature superconductors with potential in the construction of quantum computers. Some of these datasets have already been employed in the study of offline MBO methods (Kumar & Levine, 2019; Brookes et al., 2019; Fannjiang & Listgarten, 2020). However, these studies all use different sets of tasks and their evaluation protocols are highly domain-specific, making it difficult to compare across methods. In our benchmark, we incorporate modified variants of some of these datasets along with our own tasks, and provide a standardized evaluation protocol.

Recently, several methods have been proposed for specifically addressing the offline MBO problem. These methods (Kumar & Levine, 2019; Brookes et al., 2019; Fannjiang & Listgarten, 2020) typically learn models of the objective

Dataset Name	Size	Dimensions	Categories	Type	Oracle
TF Bind 8	32898	8	4	Discrete	Exact
TF Bind 10	50000	10	4	Discrete	Exact
NAS	1771	64	5	Discrete	Exact
ChEMBL	1093	31	591	Discrete	Random Forest
Superconductor	21263	86	N/A	Continuous	Random Forest
Ant Morphology	25009	60	N/A	Continuous	Exact
D'Kitty Morphology	25009	56	N/A	Continuous	Exact
Hopper Controller	3200	5126	N/A	Continuous	Exact

Table 1. **Overview of the tasks in our benchmark suite.** Design-Bench includes a variety of tasks from different domains, including several from prior work, and multiple new tasks, with both discrete and continuous design spaces, making it suitable for benchmarking offline MBO methods. In addition to the provided tasks, we explore several from prior work in Appendix D that we chose not to include in the final benchmark.

function and optionally, a generative model (Kingma & Welling, 2013; Goodfellow et al., 2014; Mirza & Osindero, 2014) of the design manifold and use them for optimization. We discuss these methods in detail in Section 6 and benchmark their performance in Section 7.

4. Design-Bench Benchmark Tasks

In this section, we describe the set of tasks included in our benchmark. An overview of the tasks is provided in Table 1. Each task in our benchmark suite comes with a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, along with a ground-truth oracle objective function f that can be used for evaluation. An offline MBO algorithm should not query the ground-truth oracle function during training, even for hyperparameter tuning. We first discuss the nature of oracles used in Design-Bench.

Expert model as oracle function. While in some of the tasks in our benchmark, such as tasks pertaining to robotics (D'Kitty Morphology, and Ant Morphology), the oracle functions are evaluated by running computer simulations to obtain the true objective values, in the other tasks, the true objective values can only be obtained by conducting expensive physical experiments. While the eventual aim of offline MBO is to make it possible to optimize designs in precisely such settings, requiring real physical experiments for evaluation makes the design and benchmarking of new algorithms difficult and time consuming. Therefore, to facilitate benchmarking, we follow the evaluation methodology in prior work (Brookes et al., 2019; Fannjiang & Listgarten, 2020) and use models built by domain experts as our ground-truth oracle functions. Note, however, that the training data provided for offline MBO is still real data – the domain expert model is used *only* to evaluate the result for benchmarking purposes. In many cases, these expert models are *also* learned, but with representations that are hand-designed, with built-in domain-specific inductive biases. The ground-truth oracle models are also trained on much more data than is made available for solving the offline MBO problem, which increases the likelihood that this expert model can provide an accurate evaluation of solu-

tions found by offline MBO, even if they lie outside the training distribution. While this approach to evaluation diminishes the realism of our benchmark since these proxy "true functions" may not always be accurate, we believe that this trade off is worthwhile to make benchmarking practical. The main purpose of our benchmark is to facilitate the evaluation and development of offline MBO algorithms, and we believe that it is important to include tasks in domains where the true objective values can only be obtained via physical experiments, which make up a large portion of the real-world MBO problems.
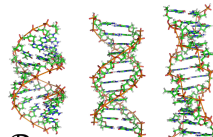
We now provide a detailed description of the tasks in our benchmark. A description of the data collection strategy and pre-processing can be found in Appendix A.

Superconductor: critical temperature maximization. The Superconductor task is taken from the domain of materials science, where the goal is to design the chemical formula for a superconducting material that has a high critical temperature. We adapt a real-world dataset proposed by Hamidieh (2018). The dataset contains 21263 superconductors annotated with critical temperatures. Prior work has employed this dataset for the study of offline MBO methods (Fannjiang & Listgarten, 2020), and we follow their convention using a random forest regression model, detailed in (Hamidieh, 2018), for our oracle. The model achieves a final Spearman's rank-correlation coefficient with a held-out validation set of 0.9210. The design space for Superconductor is a vector with 86 real-valued components representing the mixture of elements by number of atoms in the chemical formula of each superconductor.

TF Bind 8 and TF Bind 10: DNA sequence optimization.

The goal of TF Bind 8 and TF Bind 10 is to find the length-8 DNA sequence with maximum binding affinity with a particular transcription factor (`SIX6_REF_R1` by default). The ground truth binding affinities for

all 65,792 and 1,048,576 designs for the two tasks are available (Barrera et al., 2016). The design space consists of sequences of one of four categorical variables, one for each nucleotide. For TF Bind 8, we sample 32898 of all the sequences, and for TF Bind 10 we sample 50000 sequences to form the training set.

Ant and D'Kitty Morphology: robot morphology optimization.

The goal is to optimize the morphological structure of two simulated robots: Ant from OpenAI Gym (Brockman et al., 2016) and D'Kitty from ROBEL (Ahn et al., 2019). For Ant Morphology, the we need to optimize the morphology of a quadruped robot to run as fast as possible. For D'Kitty Morphology, the goal is to optimize the morphology of D'Kitty robot (shown on the right) to navigate the robot to a fixed location. Thus the goal is to find robot morphologies optimal for the given tasks. In order to control the robot with the generated morphology, we use a controller that has been optimized for the given morphology with the Soft Actor Critic algorithm (Haarnoja et al., 2018b). The morphology parameters of both robots include size, orientation, and location of the limbs, giving us 60 continuous values in total for Ant and 56 for D'Kitty. To evaluate a given design, we run robotic simulation in the MuJoCo (Todorov et al., 2012) simulator for 100 time steps, averaging 16 independent trials giving us reliable but cheap to compute estimates.
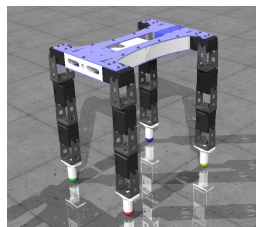
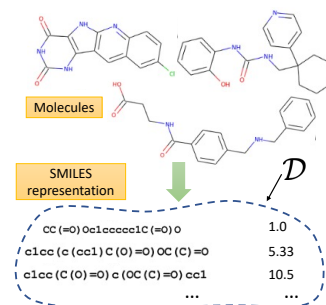**NAS: neural architecture search on CIFAR10.**

The goal of this task is to search for a good neural network architecture (Zoph & Le, 2016) to optimize the test accuracy on the CIFAR10 (Hinton et al., 2012) dataset. The model is a 32-layer convolutional neural network with residual connections, and the task requires searching over the kernel sizes and activation function types for each of the 32 layers. Given the small image size of CIFAR10, we choose the list of possible kernel sizes to be $\{2, 3, 4, 5, 6\}$. The possible choices of activation functions are ReLU, ELU, leaky ReLU, SELU (Klambauer et al., 2017) and SiLU (Elfwing et al., 2018). The combination of kernel sizes and activation functions give us a 64 dimensional discrete space with 5 categories per dimension. The
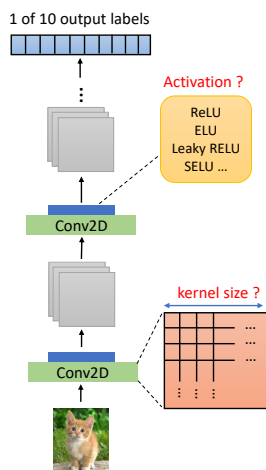
dataset is collected by randomly sampling architectures in the search space. We evaluate the design by training the produced architecture on the training CIFAR10 dataset for 20 epochs and evaluating the accuracy on the test set.

ChEMBL: molecule activity maximization for drug discovery.

The ChEMBL task in Design bench is derived from a large-scale drug property database from which the task name is derived (Gaulton et al., 2012). This database consists of pairs of molecules and assays tested for a particular chemical properties. We choose the assay whose ChEMBL id is `CHEMBL3885882` and measure its `MCHC` value. The goal of the resulting optimization problem is to design a molecule that, when paired with assay `CHEMBL3885882`, achieves a high `MCHC` value. The training set is restricted to molecules whose SMILES (Weininger, 1988) encoding has fewer than 30 tokens. This results in a training set with 1093 samples, and a design space of length 31 sequences of categorical variables that take one of 591 values.

Hopper Controller: robot neural network controller optimization. The goal in this task is to optimize the weights of a neural network policy so as to maximize the expected discounted return on the Hopper-v2 locomotion task in OpenAI Gym (Brockman et al., 2016). While this might appear similar to reinforcement learning (RL), our formulation is distinct: unlike RL, we don't have access to any form of trajectory data in the dataset. Instead, our dataset only comprises of neural network controller weights and the corresponding return values, which invalidates the applicability of conventional RL methods. We evaluate the true objective value of any design by running 1000 steps of simulation in the MuJoCo simulator conventionally ussed with this environment. The design space of this task is high-dimensional with 5126 continuous variables corresponding to the flattened weights of a neural network controller. The dataset is collected by training a PPO (Schulman et al., 2017) and recording the agent's weights every 10,000 samples.

5. Task Properties, Challenges and Considerations

The primary goal of our benchmark is to provide a general test bench for developing, evaluating, and comparing algorithms for offline MBO. While in principle any online active black-box optimization problem can be turned into an offline MBO problem by collecting a dataset of designs

and corresponding objective measurements, it is important to pick a subset of tasks that represent the challenges of real-world problems in order to convincingly evaluate algorithms and obtain insights about algorithm behavior. Therefore, several factors must be considered when choosing the tasks, which we discuss next.

Diversity and realistically challenging. First of all, the tasks need to be diverse and realistically challenging in order to prevent offline MBO algorithms from overfitting to a particular problem domain and to expect that methods performing well on this benchmark suite would also perform well on real-world offline MBO problems. Design-Bench consists of tasks that are diverse in many respects. It includes both tasks with *discrete* and with *continuous* design spaces. Continuous design spaces, equipped with metric space and ordering structures, could make the problem easier to solve than discrete design spaces. However, discrete design spaces are finite and therefore might enjoy better dataset coverage than some continuous tasks. While our tasks are not intended to directly solve real-world problems (e.g., we don't actually expect the best robot morphology in our benchmark to actually correspond to the best possible real robot morphology due to a variety of factors including limitations of the simulator), they are intended to provide designers with a representative sampling of challenges that reflect the kinds of difficulties they would face with real-world datasets, making them realistically challenging.

High-dimensional design spaces. In many real-world offline MBO problems, such as drug discovery (Gaulton et al., 2012), the design space is *high-dimensional* and good designs sparsely lie on a *thin manifold* in this high-dimensional space. This poses a challenge for many MBO methods: to be effective on such problem domains, MBO methods need to capture the thin manifold to be able to produce good designs. Prior work (Kumar & Levine, 2019) has noted that this can be very hard in practice. In our benchmark, we include a task derived from ChEMBL with up 31 dimensions and 591 categories per dimension to capture this challenge. To intuitively understand this challenge, we performed a study on some tasks in Figure 3, where we sampled 3200 designs uniformly at random from the design space and plotted a histogram of the objective values against those in the dataset we provide, which only consists of valid designs. Observe the discrepancy in objective values, where randomly sampled designs generally attain objective values lower than the best dataset sample. This suggests that performant designs only lie on a thin manifold in the design space and therefore we are very unlikely to hit a performant design by uninformed random sampling.

Highly sensitive objective function. Another important challenge that should be taken into consideration is the *high sensitivity* of objective functions, where closeness of two designs in design space need not correspond to closeness in their objective values, which may differ drastically. This challenge is naturally present in practical problems like drug discovery (Gaulton et al., 2012), where the change of a single atom could significantly alter the property of the molecule. The DKitty Morphology and Ant Morphology tasks in our benchmark suite are also particularly challenging in this respect. To visualize the high sensitivity of the objective function, we plot a one dimensional slice of the objective function around a single sample in our dataset in Figure 4. Observe that with other variables kept constant, slightly altering one variable can significantly reduce the objective value, making it hard for offline MBO methods to produce the optimal design.

Heavy-tailed data distributions. Finally, another challenging property for offline MBO methods is the shape of the data distribution. Learning algorithms are likely to exhibit poor learning behavior when the distribution of objective values in the dataset is heavy-tailed. This challenge is often present in black-box optimization (Chowdhury & Gopalan, 2019) and can hurt the performance of MBO algorithms that use a generative model as well as those that use a learned model of the objective function. As shown in Figure 3 tasks in our benchmark exhibit this heavy-tailed structure.

6. Algorithm Implementations

To provide a baseline for comparisons in future work, we benchmark a number of recently proposed offline MBO algorithms on each of our tasks. Since some of our tasks have a high input dimensionality, we chose prior methods that can handle *both* the case of offline training data (i.e., no active interaction) and high-dimensional inputs. Thus, we include MINs (Kumar & Levine, 2019), CbAS (Brookes et al., 2019), autofocusing CbAS (Fannjiang & Listgarten, 2020) and REINFORCE/CMA-ES (Williams, 1992) in our comparisons, along with a baseline naïve "gradient ascent" method that approximates the true function $f(\mathbf{x})$ with a deep neural network and then performs gradient ascent on the output of this model. In this section, we briefly discuss these algorithms, before performing a comparative evaluation in the next section. Our implementation of these algorithms are open sourced and can be found at github.com/rail-berkeley/design-baselines.

Gradient ascent (Grad). This is a simple baseline that learns a model of the objective function, $\hat{f}(\mathbf{x})$, and optimizes $\mathbf{x}$ against this learned model via gradient ascent. Formally, the optimal solution $\mathbf{x}^*$ generated by this method can be computed as a fixed point of the following update: $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \alpha \nabla_{\mathbf{x}} \hat{f}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_t}$. In practice we perform $T = 200$ gradient steps, and report $\mathbf{x}_T$ as the final solution. Such methods are susceptible to producing invalid solutions, since the learned model does not capture the manifold of
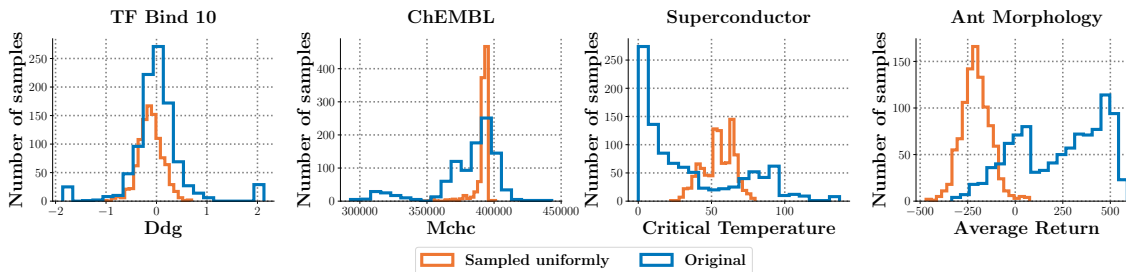
Figure 3. **Histogram (frequency distribution) of objective values in the dataset compared to a uniform re-sampling of the dataset** from the design space. In every case, re-sampling skews the distribution of values to the left, suggesting that there exists a thin manifold of valid designs in the high-dimensional design space, and most of the volume in this space is occupied by low-scoring designs. The distribution of objective values in the dataset are often heavy-tailed, for instance, in the case of ChEMBL and Superconductor.
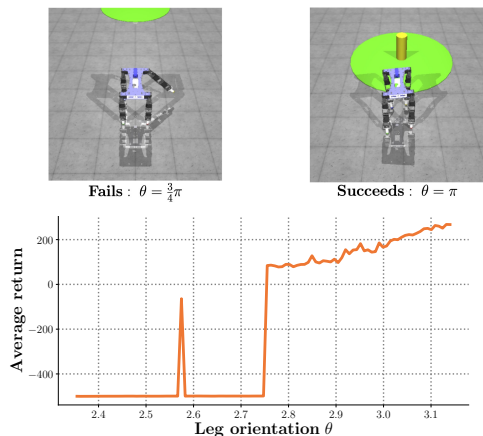

Figure 4. **Highly sensitive landscape of the ground truth objective function in DKittyMorphology.** A small change in a single dimension of the design space, for instance changing the orientation θ (x-axis) of the base of the robot's front right leg, critically impacts the performance value (y-axis). The robot's design on the left is the original D'Kitty design and is held constant while varying θ uniformly from $\frac{3}{4}\pi$ to π.

valid-designs and hence cannot constrain the resulting $\mathbf{x}_T$ to be on the manifold. We additionally evaluate a variant (**Grad. Min**) optimizing over the minimum prediction of $N = 5$ learned objective functions in an ensemble of learned objective functions and (**Grad. Mean**) that optimizes the mean ensemble prediction. We discuss additional tricks (e.g., normalization of inputs and outputs) that we found beneficial with this baseline in Appendix E.

Covariance matrix adaptation (CMA-ES). CMA-ES Hansen (2006) is a simple optimization algorithm that maintains a belief distribution over the optimal design, and gradually refines this distribution by adapting the covariance matrix using feedback from a (learned) objective function, $\hat{f}(\mathbf{x})$. Formally, let $\mathbf{x}_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ be the samples obtained from the distribution at an iteration t, then CMA-ES computes the value of learned $\hat{f}(\mathbf{x}_t)$ on samples $\mathbf{x}_t$, and fits Σ_{t+1} to the highest scoring fraction of these samples and repeats this multiple times. The learned $\hat{f}(\mathbf{x})$ is trained via supervised regression.

REINFORCE (Williams, 1992). We also evaluated a method that optimizes a learned objective function, $\hat{f}(\mathbf{x})$, using the REINFORCE-style policy-gradient estimator. REINFORCE is capable of handling non-smooth and highly stochastic objectives, making it an effective choice. This method parameterizes a distribution $\pi_\theta(\mathbf{x})$ over the design space and then updates the parameters θ of this distribution towards the design that maximizes $\hat{f}(\mathbf{x})$, using the gradient, $\mathbb{E}_{\mathbf{x} \sim \pi_\theta(\mathbf{x})}[\nabla_\theta \log \pi_\theta(\mathbf{x}) \cdot \hat{f}(\mathbf{x})]$. We train an ensemble of $\hat{f}(\mathbf{x})$ models and pick the subset of models that satisfy a validation loss threshold τ. This threshold is task-specific; for example, $\tau \leq 0.25$ is sufficient for Superconductor-v0.

Conditioning by adaptive sampling (CbAS) (Brookes et al., 2019). CbAS learns a density model in the space of design inputs, $p_0(\mathbf{x})$ that approximates the data distribution and gradually adapts it towards the optimized solution $\mathbf{x}^*$. In a particular iteration t, CbAS alternates between (**1**) training a variational auto-encoder (VAE) (Kingma & Welling, 2013) on a set of samples generated from the previous model $\mathcal{D}_t = \{\mathbf{x}_i\}_{i=1}^m; \mathbf{x}_i \sim p_{t-1}(\cdot)$ using a weighted version of the standard ELBO objective biased towards *estimated* better designs and (**2**) generating new design samples from the autoencoder to serve as $\mathcal{D}_{t+1} = \{\mathbf{x}_i | \mathbf{x}_i \sim p_t(\cdot)\}$. In order to estimate the objective values for designs sampled from the learned density model $p_t(\mathbf{x})$, CbAS utilizes separately trained models of the objective function, $\hat{f}(\mathbf{x})$ trained via supervised regression. This training process, at a given iteration t, is:

$$p_{t+1}(\mathbf{x}) := \arg\min_p \frac{1}{m} \sum_{i=1}^m \frac{p_0(\mathbf{x}_i)}{p_t(\mathbf{x}_i)} P(\hat{f}(\mathbf{x}_i) \geq \tau) \log p_t(\mathbf{x}_i)$$

$$\text{where } \{\mathbf{x}_i\}_{i=1}^m \sim p_t(\cdot). \tag{2}$$

Autofocused CbAS (Auto. CbAS) (Fannjiang & Listgarten, 2020). Since CbAS uses a learned model of the objective function $\hat{f}(\mathbf{x})$ to iteratively adapt the generative model $p(\mathbf{x})$ towards the optimized design, the function $\hat{f}(\mathbf{x})$ will inevitably be required to make predictions on shifting design distributions $p_t(\mathbf{x})$. Hence, any inaccuracy in these values can adversely affect the optimization procedure. Autofocused CbAS aims to correct for this shift by re-training

$\hat{f}(\mathbf{x})$ (now denoted $\hat{f}_t(\mathbf{x})$) under the design distribution given by the current model, $p_t(\mathbf{x})$ via importance sampling, which is then fed into CbAS.

$$\hat{f}_{t+1} := \arg\min_{\hat{f}} \quad \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \frac{p_t(\mathbf{x}_i)}{p_0(\mathbf{x}_i)} \cdot \left(\hat{f}(\mathbf{x}_i) - y_i \right)^2,$$

Model inversion networks (MINs) (Kumar & Levine, 2019). MINs learn an inverse map from the objective value to a design, $\hat{f}^{-1} : \mathcal{Y} \to \mathcal{X}$ by using objective-conditioned inverse maps, search for optimal y values during optimization and finally query the learned inverse map to produce the corresponding optimal design. MIN minimizes a divergence measure $\mathcal{L}_p(\mathcal{D}) := \mathbb{E}_{y \sim p_{\mathcal{D}}(y)} \left[D(p_{\mathcal{D}}(\mathbf{x}|y), \hat{f}^{-1}(\mathbf{x}|y)) \right]$ to train such an inverse map. During optimization, MINs obtains the optimized design by sampling from the inverse map conditioned on the optimal y-value.

Bayesian optimization (BO-qEI). We perform offline Bayesian optimization to maximize the value of a learned objective function $\hat{f}(\mathbf{x})$ by fitting a Gaussian Process, proposing candidate solutions, and labeling these candidates using $\hat{f}(\mathbf{x})$. To improve efficiency, we use the quasi-Expected-Improvement acquisition function (Wilson et al., 2017) based on the BoTorch framework (Balandat et al., 2019).

Conservative Objective Models (COMs) (Trabucco et al., 2021). COMs utilizes a single learned model of the objective function $\hat{f}(\mathbf{x}_i)$ for offline model-based optimization. COMs learns a conservative model of the objective function using an augmented regression objective that penalizes overestimation of the performance on off-manifold designs $\mathbf{x}$. Solutions are obtained by initializing $\mathbf{x}_0$ to a design from an observed training set $\mathcal{D}$, and performing T steps of gradient ascent $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \nabla_{\mathbf{x}} \alpha \hat{f}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_t}$ on the conservative objective model's predictions with respect to the design $\mathbf{x}$.

7. Benchmarking Prior Methods

In this section, we provide a comparison of prior algorithms discussed in Section 6 on our proposed tasks. For purposes of standardization, easy benchmarking, and future algorithm development, we present results for all Design-Bench tasks in Table 2. As discussed in Section 2, we allow each method to produce $K = 128$ optimized design candidates. These candidates are then evaluated with the oracle function, and we report the 100th percentile performance among them averaged over 8 independent runs, following the conventions of prior works (Fannjiang & Listgarten, 2020; Brookes et al., 2019; Kumar & Levine, 2019). We also provide unnormalized and 50th%ile results in Appendices C.3, C.2.

Algorithm setup and hyperparameter tuning. Since our goal is to generate high-performing solutions without *any* knowledge of the ground truth function, any form of hyperparameter tuning on the parameters of the learned model

should crucially respect this evaluation boundary and tuning must be performed completely offline, agnostic of the objective function. We provide a recommended method for tuning each algorithm described in Section 6 in Appendix F, which also serves as a set of guidelines for tuning future methods with similar components.

To briefly summarize, **for CbAS**, hyperparameter tuning amounts to tuning a VAE where samples from the prior distribution map to on-manifold designs after reconstruction. We empirically found that a β-VAE was essential for stability of CbAS and high values of $\beta > 1$ are especially important for modelling high-dimensional spaces. As a general task-agnostic principle for selecting β, we choose the smallest β such that the VAE's latent space does not collapse during importance sampling. Collapsing latent-spaces seem to coincide with diverging importance sampling, and the VAE's reconstructions collapsing to a single mode. **For MINs**, hyperparameter tuning amounts to fitting a good generative model. We observe that MINs is particularly sensitive to the scale of y_i when conditioning, which we resolve by normalizing the objective values. We implement MINs using WGAN-GP, and find that similar hyperparameters work well across domains. **For Gradient Ascent**, while prior works report poor performance for naïve gradient ascent optimization on top of learned models of the objective function, we find that by normalizing the designs $\mathbf{x}$ and objective values y to have unit normal statistics and scaling the learning rate as $\alpha \leftarrow \alpha\sqrt{d}$ where d is the dimension of the design space (discussed in Appendix E), a naïve gradient ascent based procedure performs reasonably well on most tasks without task-specific tuning. For discrete tasks, only the objective values are normalized, and optimization is performed over log-probabilities of designs. We then obtain optimized designs by running 200 steps of gradient ascent starting from the top scoring 128 samples in each dataset. We provide further details in Appendix F.

Results. The results for all tasks are provided in Table 2. There are several takeaways from these results. First, these results confirm that three prior offline MBO methods (MINs, CbAS, and Autofocused CbAS), are very successful at solving a wide range of offline MBO problems of varying dimensional and modality. Furthermore, perhaps somewhat surprisingly, a classical CMA-ES baseline is competitive with several highly sophisticated MBO methods in 4 out of 8 tasks (Table 2). This result suggests that it might be difficult for generative models to capture high-dimensional task distributions with enough precision to be used for optimization, and in a number of tasks, these components might be unnecessary. Additionally a naive gradient ascent baseline is competitive with complex approaches utilizing generative modelling on 4 of the 8 tasks. However, on the other hand, as described in Appendix E and F.4, baseline is also sensitive to certain design choices such as input normalization
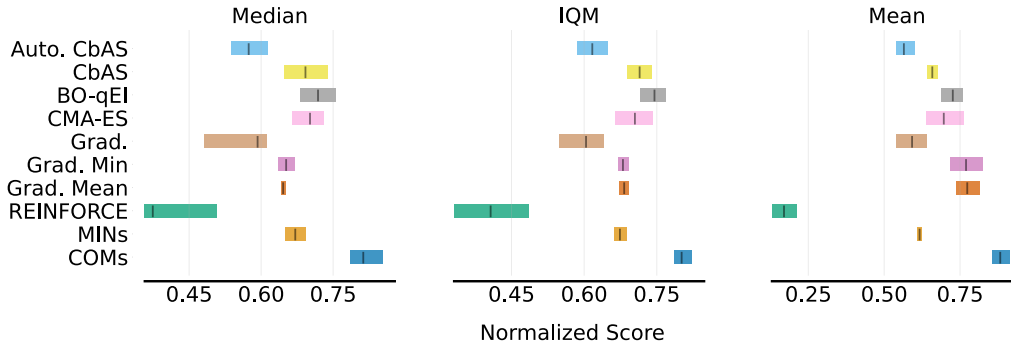
Figure 5. **Median, IQM and mean** (Agarwal et al., 2021) aggregated 100th percentile normalized scores (with 95% Stratified Bootstrap CIs) for the tasks in Design-Bench.

		TF Bind 8	TF Bind 10	ChEMBL	NAS	Superconductor	Ant Morph.	DKitty Morph.	Hopper
$\mathcal{D}$ (**best**)		0.439	0.467	0.605	0.436	0.400	0.565	0.884	1.0
Auto. CbAS		0.910 ± 0.044	0.630 ± 0.045	0.249 ± 0.305	0.506 ± 0.074	0.421 ± 0.045	0.882 ± 0.045	0.906 ± 0.006	0.137 ± 0.005
CbAS		0.927 ± 0.051	0.651 ± 0.060	0.473 ± 0.264	0.683 ± 0.079	0.503 ± 0.069	0.876 ± 0.031	0.892 ± 0.008	0.141 ± 0.012
BO-qEI		0.798 ± 0.083	0.652 ± 0.038	0.596 ± 0.226	1.079 ± 0.059	0.402 ± 0.034	0.819 ± 0.000	0.896 ± 0.000	0.550 ± 0.118
CMA-ES		0.953 ± 0.022	0.670 ± 0.023	0.085 ± 0.225	0.985 ± 0.079	0.465 ± 0.024	1.214 ± 0.732	0.724 ± 0.001	0.604 ± 0.215
Grad.		0.977 ± 0.025	0.657 ± 0.039	0.307 ± 0.308	0.433 ± 0.000	0.518 ± 0.024	0.293 ± 0.023	0.874 ± 0.022	1.035 ± 0.482
Grad. Min		0.984 ± 0.012	0.649 ± 0.032	0.653 ± 0.024	0.433 ± 0.000	0.506 ± 0.009	0.479 ± 0.064	0.889 ± 0.011	1.391 ± 0.589
Grad. Mean		0.986 ± 0.012	0.645 ± 0.018	0.652 ± 0.005	0.433 ± 0.000	0.499 ± 0.017	0.445 ± 0.080	0.892 ± 0.011	1.586 ± 0.454
REINFORCE		0.948 ± 0.028	0.663 ± 0.034	0.164 ± 0.285	-1.895 ± 0.000	0.481 ± 0.013	0.266 ± 0.032	0.562 ± 0.196	-0.020 ± 0.067
MINs		0.905 ± 0.052	0.616 ± 0.021	0.000 ± 0.000	0.717 ± 0.046	0.499 ± 0.017	0.445 ± 0.080	0.892 ± 0.011	0.424 ± 0.166
COMs		0.973 ± 0.016	0.730 ± 0.136	0.633 ± 0.000	0.459 ± 0.139	0.439 ± 0.033	0.944 ± 0.016	0.949 ± 0.015	2.056 ± 0.314

Table 2. **100th percentile** evaluations. Results are averaged over 8 trials, and ± indicates the standard deviation of the performance. The objective value normalization procedure is described in Appendix C.1.

schemes and the number of optimization steps T. Therefore, while not a full-fledged offline MBO method, we believe that gradient ascent has potential to form a fundamental building block for future offline MBO methods.

Finally, we remark that the performance of methods in Table 2 differ from the those reported by prior works. This difference stems from the standardization procedure employed in dataset generation (which we discuss in Appendix A).

8. Discussion and Conclusion

Offline model-based optimization carries the promise to convert existing databases of designs into powerful optimizers, without the need for expensive real-world experiments for actively querying the ground truth objective function. However, due to the lack of standardized benchmarks and evaluation protocols, it has been difficult to accurately track the progress of offline MBO methods. To address this problem, we introduce Design-Bench, a benchmark suite of offline MBO tasks that covers a wide variety of domains, and both continuous and discrete, low and high dimensional design spaces. We provide a comprehensive evaluation of existing methods under identical assumptions. The comparatively high efficacy of even simple baselines such as CMA-ES and naïve gradient ascent suggests the need for careful tuning and standardization of methods in this area. An interesting avenue for future work in offline MBO is to devise methods

that can be used to perform model and hyperparameter selection. One promising approach to address this problem is to devise methods for offline evaluation of produced solutions. We hope that our benchmark will be adopted as the standard metric in evaluating offline MBO algorithms and provides insight in future algorithm development.

Acknowledgements

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

Ahn, M., Zhu, H., Hartikainen, K., Ponte, H., Gupta, A., Levine, S., and Kumar, V. ROBEL: RObotics BEnchmarks for Learning with low-cost robots. In *Conference on Robot Learning (CoRL)*, 2019.

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*, 2020. URL https://openreview.net/forum?id=HklxbgBKvr.

Angermüller, 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. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 324–334. PMLR, 2020. URL http://proceedings.mlr.press/v119/angermueller20a.html.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. Botorch: Programmable bayesian optimization in pytorch. *CoRR*, abs/1910.06403, 2019. URL http://arxiv.org/abs/1910.06403.

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.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

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

Chowdhury, S. R. and Gopalan, A. Bayesian optimization under heavy-tailed payoffs. 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. 13790–13801, 2019. URL http://papers.nips.cc/paper/9531-bayesian-optimization-under-heavy-tailed-payoffs.

Dodge, J., Gururangan, S., Card, D., Schwartz, R., and Smith, N. A. Show your work: Improved reporting of experimental results. In Inui, K., Jiang, J., Ng, V., and Wan, X. (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 2185–2194.

Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1224. URL https://doi.org/10.18653/v1/D19-1224.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

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 *International Conference on Learning Representations*, 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. *arXiv preprint arXiv:2004.07219*, 2020.

Gaulton, A., Bellis, L. J., Bento, A. P., Chambers, J., Davies, M., Hersey, A., Light, Y., McGlinchey, S., Michalovich, D., Al-Lazikani, B., and Overington, J. P. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(Database issue): D1100–D1107, Jan 2012. ISSN 1362-4962. doi: 10.1093/nar/gkr777. URL https://pubmed.ncbi.nlm.nih.gov/21948594. 21948594[pmid].

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

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*, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

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.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Jordan, S. M., Chandak, Y., Cohen, D., Zhang, M., and Thomas, P. S. Evaluating the performance of reinforcement learning algorithms. *CoRR*, abs/2006.16958, 2020. URL https://arxiv.org/abs/2006.16958.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *Proceedings of the 31st international conference on neural information processing systems*, pp. 972–981, 2017.

Kolda, T. G., Lewis, R. M., and Torczon, V. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

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

Lizotte, D. J. *Practical bayesian optimization*. University of Alberta, 2008.

Mirza, M. and Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Pal, S. K., Rai, C., and Singh, A. P. Comparative study of firefly algorithm and particle swarm optimization for noisy non-linear optimization problems. *International Journal of intelligent systems and applications*, 4(10):50, 2012.

Perrone, V., Jenatton, R., Seeger, M., and Archambeau, C. Multiple adaptive bayesian linear regression for scalable bayesian optimization with warm start. *arXiv preprint arXiv:1712.02902*, 2017.

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. and Kroese, D. P. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

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.

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 2016. 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 (1):148–175, 2015.

Shen, W.-J., Wong, H.-S., Xiao, Q.-W., Guo, X., and Smale, S. Introduction to the peptide binding problem of computational immunology: New results. *Foundations of Computational Mathematics*, 14(5):951–984, Oct 2014. ISSN 1615-3383. doi: 10.1007/s10208-013-9173-9. URL https://doi.org/10.1007/s10208-013-9173-9.

Sivaprasad, P. T., Mai, F., Vogels, T., Jaggi, M., and Fleuret, F. On the tunability of optimizers in deep learning. *CoRR*, abs/1910.11758, 2019. URL http://arxiv.org/abs/1910.11758.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

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 *International conference on machine learning*, pp. 2171–2180, 2015.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Trabucco, B., Kumar, A., Geng, X., and Levine, S. Conservative objective models for effective offline model-based optimization. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10358–10368. PMLR, 2021. URL http://proceedings.mlr.press/v139/trabucco21a.html.

Van Laarhoven, P. J. and Aarts, E. H. Simulated annealing. In *Simulated annealing: Theory and applications*, pp. 7–15. Springer, 1987.

Weininger, D. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

Whitley, D. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

Williams, C. K. and Rasmussen, C. E. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

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

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.

Yang, X.-S. and Slowik, A. Firefly algorithm. In *Swarm Intelligence Algorithms*, pp. 163–174. CRC Press, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Appendices

A. Data Collection

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

A.1. TF Bind 8 and TF Bind 10

The TF Bind 8 and TF Bind 10 tasks are derivativesof the transcription factor binding activity survey performed by (Barrera et al., 2016), where the binding activity scores of every possible 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., 2020; Angermüller et al., 2020). TF Bind 8 is a fully characterized dataset containing 65792 samples, representing every possible length 8 combination of nucleotides $x_{\text{TFBind8}} \in \{0, 1\}^{8 \times 4}$. The training set given to offline MBO algorithms is restricted to the bottom 50%, which results in a visible training set of 32898 samples.

A.2. ChEMBL

The ChEMBL task is a derivative of a much larger dataset that is derived from ChEMBL (Gaulton et al., 2012), a large database of chemicals and their properties. The datawas originally collected by performing physical experiments on a large number of molecules, and measuring a chemical property in the presence of a target assay. We have processed the ChEMBL database— available at `https://www.ebi.ac.uk/chembl/g/#browse/activities`—into collections of smaller datasets mapping particular molecules to measured values, determined by a target assay that accompanies each set. We choose the assay specified by `ASSAY_CHEMBL_ID =` `CHEMBL3885882` and select the standard type of `MCHC` as the measurement to maximize with offline model-based optimization. The resulting dataset has 1093 samples in total. This assay is chosen for its high validation rank correlation, namely 0.7141, when fitting a random forest regression model to map molecules to `MCHC` values. The majority of other assays in ChEMBL produce a validation rank correlation below 0.5. We preprocess the dataset by converting each molecule into a SMILES string using RDKit, and then apply the DeepChem `SmilesTokenizer` to convert each SMILES string into a sequence of integer tokens. We then remove all molecules whose SLIMES sequence is longer than a maximum of 31 tokens with the vocabulary has 591 elements, $x_{\text{ChEMBL}} \in \{0, 1\}^{31 \times 591}$. When evaluating MBO methods, we remove the top 50% of molecules sorted by their `MCHC` value to increase task difficulty.

A.3. 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 $x_{\text{Superconductor}} \in \mathcal{R}^{86}$. 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.

A.4. Ant & D'Kitty Morphology

Both morphology tasks are collected by us, and share methodology. The goal of these tasks is to design the morphology of a quadrupedal robot—an ant or a D'Kitty—such that the agent is able to crawl quickly in a particular direction. In order to collect data for this environment, we create variants of the MuJoCo Ant and the ROBEL D'Kitty agents that have parametric morphologies. The goal is to determine a mapping from the morphology of the agent to the average return of the agent using a controller optimized for that morphology. In order to facilitate fast optimization, we pre-compute a morphology conditioned neural network controller using SAC (Haarnoja et al., 2018a) that has been trained to perform optimally on a wide range of morphologies. For both the Ant and the D'Kitty, we train the controllers for more than ten million environment steps, and a maximum episode length of 200, with all other settings as default. These morphology conditioned controllers are

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 controllers, 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 morphology-conditioned controllers. 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 restarted, until 25000 morphologies were collected, resulting in 25009 samples for both the Ant and D'Kitty. The design space for Ant morphologies is $x_{Ant} \in \mathcal{R}^{60}$, whereas for D'Kitty morphologies is $x_{D'Kitty} \in \mathcal{R}^{56}$. We sremove the top 40% of samples when training offline MBO algorithms.

A.5. NAS

The data for the NAS task is collected by us. The goal of this task is to search for a good neural network architecture to optimize the test accuracy on the CIFAR10 dataset. The architecture search space is a 64-dimensional discrete variable with 5 categories for each dimension. We collect the dataset by randomly sample architecture designs in the search space, and train them on the CIFAR10 dataset. We sample 2440 total designs, and select the bottom performing 70% to be our training set. This gives us 1771 samples in total, with the test accuracy ranging from 59.3% to 63.8%.

A.6. Hopper Controller

The goal of this task is to design a set of weights for as neural network policy, in order to achieve high expected return when evaluating that policy. The data collected for Hopper Controller 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 $x_{Hopper} \in \mathcal{R}^{5126}$. 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 objective score values 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.

B. Oracle Functions

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

B.1. TF Bind 8 and TF Bind 10

TF Bind 8 and TF Bind 10 are a fully characterized discrete offline MBO tasks, which means that all possible designs have been evaluated (Barrera et al., 2016) and are contained in the full hidden datasets. The oracles are therefore implemented simply as a lookup table that returns the score corresponding to a particular 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 algorithm during training.

B.2. ChEMBL

We tested several models as candidate oracle functions for ChEMBL (Gaulton et al., 2012), including a Gaussian Process, Random Forest, CNN, and Transformer regression models. We ultimately chose the Random Forest model in scikit-learn due to its quick inference and relatively high performance compared with neural network alternatives, achieving a spearman's rank correlation coefficient of 0.7141 with a held-out validation set. These models were trained on the entire hidden ChEMBL dataset for `ASSAY_CHEMBL_ID = CHEMBL3885882` with standard type `MCHC` encoded into SMILES and tokenized. Hyperparameters for the random forest oracle are provided in the official github release of design-bench.

B.3. 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 rank correlation of 0.9155 with a held-out validation set.

B.4. Ant & D'Kitty Morphology

The Ant & D'Kitty Morphology 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 the reinforcement learning algorithm SAC for 10 million steps for each task, and are ReLU networks with two hidden layers of size 64.

B.5. NAS

The NAS task in the design bench uses an exact oracle, where we train the proposed architecture on CIFAR10 and then test it on the test set. To perform the evaluation, we construct the proposed architecture using PyTorch, and train it for 20 epochs using batch size 256 and then compute the test accuracy on the test set.

B.6. Hopper Controller

Unlike the previously described tasks, Hopper Controller implements an exact oracle function. For Hopper Controller the oracle takes the form of a single rollout using the Hopper-v2 MuJoCo environment. The designs for Hopper Controller 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.

C. Experimental Details

In this section we present additional details for the experiments, including the score normalization process and 50th percentile performance.

C.1. Objective Normalization

In order to report performance on the same order of magnitude for each offline model-based optimization task in Design-Bench, we normalize the performance reported in Table 2 by calculating the minimum objective value $y_{\min}$ and the the maximum objective value $y_{\max}$ in the full unobserved dataset associated with each offline model-based optimization problem. *Crucially*, note that this is not the

same as normalizing with respect to the best and worst samples in the training dataset used by the offline MBO algorithm, but rather a bigger dataset of designs and objective values. We then report performance by calculating what fraction of the distance between $y_{\min}$ and $y_{\max}$ is attained by a particular offline MBO baseline.

$$y_{\text{normalized}}(y) = \frac{y - y_{\min}}{y_{\max} - y_{\min}} \qquad (3)$$

The final performance $y_{\text{normalized}}$ is the normalized performance of an offline MBO method that achieved an unprocessed objective value of y. The result is larger than one when the offline MBO method finds a solution more performance than all solutions in the full unobserved dataset associated with the corresponding task. The result is less than zero when the offline MBO method finds a solution attaining less performance than all samples in the full unobserved dataset.

C.2. 50th Percentile Experiment Results

In this section, we present the 50th percentile performance of the runs presented in main paper in Table 2. Similar to the 100th percentile performance reported in the main text, performance is calculated by evaluating solutions to each task found by an optimization method, subtracting the minimum objective value present in the corresponding task dataset, and dividing by the range of objective values present in the corresponding task dataset. The result is a performance of greater than one if optimization converges to a solution with a higher objective value that the best observed design in the corresponding task dataset.

C.3. Unnormalized Experimental Results

In this section, we present the raw 100th percentile performance of the runs presented in main paper in Table 2. These values, presented in Table 4, represent the mean raw objective values and the standard deviation of the objective values attained by various offline MBO methods.

C.4. Computation Resources

The amount of computation resources required to produce the experiments in this paper is relatively modest except for the NAS tasks. We ran our experiments on a single server with 2 Intel Xeon E5-2698 v4 CPUs and 8 Nvidia Tesla V100 GPUs. All our experiments can be completed within 96 hours on this single machine.

D. Additional MBO Tasks That Were Discarded From Our Benchmark

The main benchmark consists of eight offline MBO tasks, four of which have discrete design-spaces, and four of which have contiguous design-spaces. In addition to the provided

	TF Bind 8	TF Bind 10	ChEMBL	NAS	Superconductor	Ant Morphology	D'Kitty Morphology
Auto. CbAS	0.419 ± 0.007	0.461 ± 0.007	-1.823 ± 0.000	0.217 ± 0.005	0.131 ± 0.010	0.364 ± 0.014	0.736 ± 0.025
CbAS	0.428 ± 0.010	0.463 ± 0.007	-1.807 ± 0.004	0.292 ± 0.027	0.111 ± 0.017	0.384 ± 0.016	0.753 ± 0.008
BO-qEI	0.439 ± 0.000	0.467 ± 0.000	-1.774 ± 0.020	0.544 ± 0.099	0.300 ± 0.015	0.567 ± 0.000	0.883 ± 0.000
CMA-ES	0.537 ± 0.014	0.484 ± 0.014	-1.763 ± 0.019	0.591 ± 0.102	0.379 ± 0.003	-0.045 ± 0.004	0.684 ± 0.016
Gradient Ascent	0.609 ± 0.019	0.474 ± 0.005	-1.772 ± 0.018	0.433 ± 0.000	0.476 ± 0.022	0.134 ± 0.018	0.509 ± 0.200
Grad. Min	0.645 ± 0.030	0.470 ± 0.002	-1.769 ± 0.014	0.433 ± 0.000	0.471 ± 0.016	0.185 ± 0.008	0.746 ± 0.034
Grad. Mean	0.616 ± 0.023	0.471 ± 0.004	-1.757 ± 0.010	0.433 ± 0.000	0.469 ± 0.022	0.187 ± 0.009	0.748 ± 0.024
MINs	0.421 ± 0.015	0.468 ± 0.006	-1.745 ± 0.000	0.433 ± 0.000	0.336 ± 0.016	0.618 ± 0.040	0.887 ± 0.004
REINFORCE	0.462 ± 0.021	0.475 ± 0.008	-1.805 ± 0.003	-1.895 ± 0.000	0.463 ± 0.016	0.138 ± 0.032	0.356 ± 0.131
COMs	0.497 ± 0.038	0.465 ± 0.008	0.633 ± 0.000	0.287 ± 0.173	0.386 ± 0.018	0.519 ± 0.026	0.885 ± 0.003

Table 3. **50th percentile** evaluations for baselines on every task. Results are averaged over 8 trials, and the ± indicates the standard deviation of the reported performance. This table corresponds to the normalized performance, using the normalization methodology described in Appendix C.1

.

	TF Bind 8	TF Bind 10	ChEMBL	NAS	Superconductor	Ant Morphology	D'Kitty Morphology
Auto. CbAS	0.910 ± 0.044	0.655 ± 0.178	42467.285 ± 0.000	64.530 ± 0.764	77.910 ± 8.361	474.888 ± 44.424	226.156 ± 7.043
CbAS	0.927 ± 0.051	0.738 ± 0.239	46681.988 ± 4987.456	66.360 ± 0.820	93.078 ± 12.695	469.499 ± 30.570	209.412 ± 9.593
BO-qEI	0.798 ± 0.083	0.742 ± 0.150	30069.684 ± 3187.300	70.447 ± 0.606	74.322 ± 6.347	413.084 ± 0.000	213.816 ± 0.000
CMA-ES	0.953 ± 0.022	0.811 ± 0.090	31607.031 ± 1578.222	69.475 ± 0.821	86.072 ± 4.508	799.394 ± 715.702	4.290 ± 1.505
Gradient Ascent	0.977 ± 0.025	0.762 ± 0.155	32514.541 ± 2612.903	63.770 ± 0.000	95.789 ± 4.436	-100.265 ± 22.118	187.206 ± 27.274
Grad. Min	0.984 ± 0.012	0.729 ± 0.126	32617.006 ± 370.390	63.770 ± 0.000	93.590 ± 1.719	80.853 ± 62.308	205.639 ± 13.427
Grad. Mean	0.986 ± 0.012	0.714 ± 0.071	33715.059 ± 1136.034	63.770 ± 0.000	92.265 ± 3.206	48.064 ± 78.555	209.355 ± 13.928
MINs	0.905 ± 0.052	0.599 ± 0.082	42732.578 ± 5126.862	66.709 ± 0.471	86.702 ± 4.171	505.515 ± 34.934	273.479 ± 14.184
REINFORCE	0.948 ± 0.028	0.786 ± 0.137	41448.012 ± 3220.380	39.720 ± 0.000	88.996 ± 2.389	-127.440 ± 30.831	-194.540 ± 238.857
COMs	0.945 ± 0.033	0.649 ± 0.153	391827.500 ± 2273.631	64.041 ± 1.431	81.238 ± 6.170	535.125 ± 16.064	278.344 ± 17.727

Table 4. **Unnormalized 100th percentile** unnormalized evaluations for baselines on every task. Results are averaged over 8 trials, and the ± indicates the standard deviation of the reported performance. This table corresponds to the unnormalized performance.

tasks, we also experimented with two other candidate MBO tasks from prior work (Angermüller et al., 2020; Brookes et al., 2019), but chose to not include them in the final benchmark due to lack in-distinguishable results across all methods, suggesting that these tasks may not be suitable for devising better algorithms.

D.1. GFP

GFP uses the oracle function derived from Rao et al. (2019). This oracle is a transformer regression model with 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. Our Transformer has a Spearman's rank correlation coefficient of 0.8497 with a held-out validation set derived from the GFP dataset.

The GFP task provided is a derivative of the GFP dataset (Sarkisyan et al., 2016). 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 $\mathbf{x}_{GFP} \in \{0, 1\}^{237 \times 20}$. 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 for-

mat 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 prior work (Brookes et al., 2019).

D.2. UTR

UTR uses a Transformer as the oracle function, which differs from the CNN that was originally used by (Angermüller et al., 2020). Our reasoning for making this change is that the Transformer 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). This Transformer has 4 attention blocks and a hidden size of 64. The Transformer 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 model has a spearman's rank correlation of 0.6424 with a held-out validation set.

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 Angermüller et al. (2020), 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 $\mathbf{x}_{UTR} \in \{0, 1\}^{50 \times 4}$ 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.3. Additional Experimental Results

We report the normalized performance of all baselines on the three additional MBO tasks that were not chosen for inclusion in the benchmark. Note that for GFP (Brookes et al., 2019) and UTR (Angermüller et al., 2020; Sample et al., 2019) performance of offline MBO method is not not distinguishable, and we consider this an indication each task is not suitable for benchmarking offline MBO methods. We encourage future revisions of these tasks.

	GFP	UTR
Auto. CbAS	0.865 ± 0.000	0.691 ± 0.012
CbAS	0.865 ± 0.000	0.694 ± 0.010
BO-qEI	0.254 ± 0.352	0.684 ± 0.000
CMA-ES	0.054 ± 0.002	0.707 ± 0.014
Grad.	0.864 ± 0.001	0.695 ± 0.013
Grad. Min	0.864 ± 0.000	0.696 ± 0.009
Grad. Mean	0.864 ± 0.000	0.693 ± 0.010
MINs	0.865 ± 0.001	0.697 ± 0.010
REINFORCE	0.865 ± 0.000	0.688 ± 0.010
COMs	0.864 ± 0.000	0.699 ± 0.011

Table 5. **Normalized 100th percentile** normalized evaluations for baselines on unused tasks. Each entry reports the empirical mean and empirical standard deviation over 8 independent trials.

E. Normalization Of Inputs and Outputs Is Important for Gradient Ascent

An important component for the good performance of the gradient ascent baseline is the normalization of design space. We found that the identical gradient-ascent baseline performed a factor 1.4x worse on Hopper Controller, when optimizing in the space of unnormalized designs and objective values, as seen in Figure E. This indicates that normalization is key in obtaining good performance with a naïve gradient ascent baseline. For continuous design-space tasks, we normalize both the designs, and the scores to have unit Gaussian statistics. For discrete design-space tasks, we first
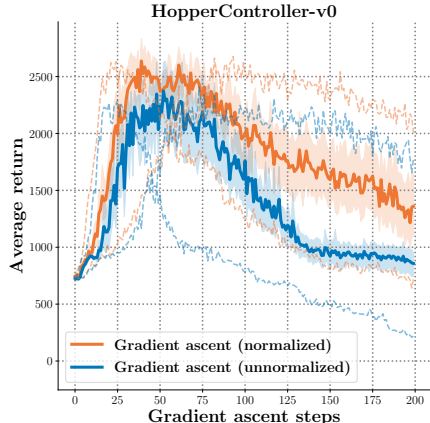
map designs to real-valued logits of a categorical distribution before performing this normalization. See the official code for how this mapping is performed. This is a necessary part of the optimization workflow because scores vary by several orders of magnitude in the dataset, for example, 0.91 for TF Bind 8 and as high as 799.394 for Ant Morphology. The specific normalization equation is given below.

$$\tilde{\mathbf{x}}_{i,j} = \frac{\mathbf{x}_{i,j} - \mu(\mathbf{x}_{\cdot,j})}{\sigma(\mathbf{x}_{\cdot,j})} \quad : \quad \mathbf{x} \in \mathbb{R}^{N \times D} \qquad (4)$$

We also normalize the objective values in a similar fashion to have unit Gaussian statistics. The result in a new set of designs $\tilde{\mathbf{x}}$ and objective values $\tilde{y}$ that is optimized over

$$\tilde{y}_{i,j} = \frac{y_{i,j} - \mu(y_{\cdot,j})}{\sigma(y_{\cdot,j})} \quad : \quad y \in \mathbb{R}^{N \times 1} \qquad (5)$$

The gradient ascent procedure is performed in the space of these normalized designs. Suppose T steps of gradient ascent have been taken, and a final normalized solution $\tilde{\mathbf{x}}_T^*$ is found. This solution is de-normalized using the following transformation.

$$(\mathbf{x}_T^*)_{ij} = (\tilde{\mathbf{x}}_T^*)_{ij} \cdot \sigma(\mathbf{x}_{\cdot,j}) + \mu(\mathbf{x}_{\cdot,j}) \qquad (6)$$

This normalization strategy is heavily inspired by data whitening, which is known to reduce the variance of machine learning algorithms that learn discriminative mappings on that data. The learned model of the objective function is one such discriminative model, and normalization likely improves the consistency of Gradient Ascent across independent experimental trials.

F. Hyperparameter Selection Workflow

Hyperparameter tuning under a restricted computational budget is emerging as an import research domain in optimization (Sivaprasad et al., 2019; Dodge et al., 2019; Jordan et al., 2020). Care must be taken when tuning each of the prescribed algorithms so that only offline information about

the task is used for hyperparameter selection. Formally, this means that the hyperparameters, $\mathcal{H}$, are conditionally independent of the particular value of the performance metric $\mathcal{M}$, given the offline task dataset $\mathcal{D}$. Examples of hyperparameter selection strategies that violate this requirement might, for example, perform a grid search over $\mathcal{H}$ and take the set that maximizes the performance metric, but this is not offline. An example of a tuning strategy that is fully offline is tuning the parameters of a learned model such that is is a good fit for the task dataset $\mathcal{D}$. One can choose $\mathcal{H}$ that minimizes a validation loss, such as negative log likelihood. A detailed record of hyperparameters can be found in the experiment scripts located alongside our reference implementations: .

We now present guidelines for hyperparameter selection (i.e. **workflow**) for methods evaluated in the benchmark. These are general principles that can be used to tune the hyperparameters of these methods on a new task in an offline fashion. While we only present workflow details for methods we benchmark in Section 7, we expect that these general strategies will allow users to devise analogous schemes for tuning hyperparameters of new offline MBO methods with shared components.

F.1. Strategy For Autofocused CbAS

The main tunable components of Autofocused methods (Fannjiang & Listgarten, 2020) are the learned objective function, and the generative model fit to the data distribution. When training the learned objective function, tracking a validation performance metric like rank correlation is helpful to ensure that the resulting learned model is able to generalize beyond its training dataset. This tracking is especially important for Autofocused methods because re-fitting the learned objective model during importance can lead to divergence if the importance weights generated by Autofocusing are very large or very small in magnitude. The algorithm is tuned well if, for example, the validation rank correlation stays above a positive threshold, such as a threshold of 0.9.

The second component of Autofocused methods is the fit of the generative model used for sampling designs. The algorithm has the best chance of success if the generative model can generalize beyond the dataset in which it was trained. This can be monitored by holding out a validation set and tracking a metric such as negative log likelihood on this held-out set. In the case when the generative model is not an exact likelihood-based generative model—for example, a VAE—other validation metrics can be used that measure the fit of the generative model on a validation set. The generative model is especially impacted by the importance sampling procedure used by Estimation of Distribution Algorithms (EDAs), and tracking the effective sample size of

the importance weights can help diagnose when the generative model is failing to generalize to a validation set.

F.2. Strategy For CbAS

The main tunable components of CbAS methods (Brookes et al., 2019) are the learned objective function, and the generative model fit to the data distribution. While the learned objective function is not affected by the importance sampling weights generated by CbAS, the same tuning strategy described in section F.1 that focuses on generalization to a validation set is effective. Generative model tuning can also follow an identical strategy to that described in section F.1, which focuses on the ability for the generative model to represent samples outside of its training set. In the case of a β-VAE, which is used with CbAS in this work, the main parameter for controlling this generalization ability is the β parameter. We found that β is task specific, and must be found in order for the CbAS optimizer using β-VAE to generate samples that are in the same distribution as its validation set. This value can be tuned in practice using a validation metric like that in section F.1.

F.3. Strategy For MINs

The main tunable components of MINs (Kumar & Levine, 2019) are the learned objective function, and the generative model fit to the data distribution. The learned objective function is typically trained using a maximum likelihood objective, and the validation log-likelihood (or regression error) can be directly tracked. The learned objective function should train until a minimum validation loss is reached, which ensured that the model will generalize as well as possible beyond its training set. Since only the static task dataset is used for this—it may be split into train/validation sets—this tuning strategy is fully offline.

The generative model for MINs is an inverse mapping $\mathbf{x} = f^{-1}(y, \mathbf{z})$, conditioned on the objective value y. Training conditional generative models is considerable less stable than unconditional generative models, so in addition to monitoring the fit of a validation set recommended in section F.1, it is also necessary to track the extent of the dependence of the generative model's predictions on the objective value y. This can be evaluated in practice by comparing the distribution of x from the conditional generative model $p(\mathbf{x}|y)$ to an unconditional generative model $p(\mathbf{x})$ with an identical initialization, or by comparing if $p(\mathbf{x}|y)$ is independent of y by querying the inverse model for different values of y and visualizing the similarity in the predictions of $\mathbf{x}$. One metric for more formally studying the extent of the dependence of $\mathbf{x}$ on $\mathbf{z}$ is the mutual information $I(\mathbf{x}; \mathbf{z})$. The conditional generative model has an appropriate fit if for some positive threshold c we have that $I(\mathbf{x}; \mathbf{z}) > c$.

F.4. Strategy For Gradient Ascent

The main tunable components of Gradient Ascent MBO methods are the learned objective function, and the parameters for gradient ascent. The learned objective function is typically trained using a maximum likelihood objective under a Gaussian distribution, and the methodology for obtaining a high-performing learned objective function is identical to that in section F.3. The second aspect of gradient ascent MBO algorithms are the parameters of the gradient-based optimizer for the designs—such as its learning rate, and the number of gradient steps it performs. The learning rate should be small enough that the gradient steps taken increase the prediction of the learned objective function—if the learning rate is too large, gradient steps may not follow the path of steepest ascent of the objective function. The number of gradient steps is more difficult to tune. The strategy we used is a fixed number of steps, and an offline criterion to select this parameter is future work.

F.5. Strategy For REINFORCE

The main tunable components of REINFORCE-based MBO methods are the learned objective function, and the parameters for the policy gradient estimator. The learned objective function is typically trained using a maximum likelihood objective, and the methodology for obtaining a high-performing learned objective function is identical to that in section F.3. The remaining parameters to tune are specific to REINFORCE. The distribution of the policy should be carefully selected to be able to model the distribution of designs. For continuous MBO tasks, a Gaussian distribution is appropriate, and for discrete MBO tasks, a categorical distribution is appropriate. In addition, the learning rate, and optimizer should be selected so that policy updates improve the model-predicted score.

F.6. Strategy For Bayesian Optimization

The main tunable components of Bayesian Optimization MBO methods (Balandat et al., 2019) are the learned objective function, and the parameters for the bayesian optimization loop. The learned objective function is typically trained using a maximum likelihood objective, and the methodology for obtaining a high-performing learned objective function is identical to that in section F.3. For a detailed review of the strengths and weaknesses of various Bayesian Optimization strategies and their hyperparameters, we refer the reader to the BoTorch documentation, available at the BoTorch website `https://botorch.org/docs/overview`. In this work we employ a Gaussian Process as the model, and the quasi-Monte Carlo Expected Improvement acquisition function, which has the advantage of scaling up to our high-dimensional optimization problems.

F.7. Strategy For Covariance Matrix Adaptation (CMA-ES)

The main tunable components of Covariance Matrix Adaptation MBO methods are the learned objective function, and the parameters for the evolution strategy. The learned objective function is typically trained using a maximum likelihood objective, and the methodology for obtaining a high-performing learned objective function is identical to that in Subsection F.3. For a detailed review of the strengths and weaknesses of various Bayesian Optimization strategies and their hyperparameters, we refer the reader to an open-source implementation of CMA-ES and its corresponding documentation `https://github.com/CMA-ES/pycma`. In this work we employ the default settings for CMA-ES reported in this open source implementation, with $\sigma = 0.5$.

F.8. Strategy For Conservative Objective Models (COMs)

Conservative Objective Models has three main tunable parameters, and we refer the reader to the original paper for a full experimental description (Trabucco et al., 2021). The first parameter for COMs is the degree to which the objective model is allowed to overestimate the objective value for off-manifold designs. This parameter can be implemented as a constraint with threshold τ, or as a penalty with weight α. This parameter is chosen to be as high as possible, permitting high validation performance. When either τ or α imposes too much conservatism, this regularizes the objective model, and may lead the model to poorly fit the dataset $\mathcal{D}$. This parameter is uniformly chosen to be 2 for all discrete tasks and 0.5 for all continuous tasks. The second tunable parameter of COMs is the number of gradient ascent steps to perform when optimizing $\mathbf{x}$, and is uniformly chosen to be 50. The final parameter is the learning rate used when optimizing $\mathbf{x}$, which is uniformly chosen to be $2\sqrt{d}$ for all discrete tasks and $0.05\sqrt{d}$ for all continuous tasks, where d is the cardinality of the design space.