Appendices

A. Method Details

A.1. Ablation of τ

Equation 5 introduces a hyperparameter τ, that controls how conservative the function $\hat{f}_\theta(\mathbf{x})$ is during training. In this section, we provide an ablation study showing the sensitivity of COMs to this hyperparameter τ. In this experiment, in order to demonstrate the relationship between the parameter τ, and the stability of performance more clearly, we generate adversarial samples $\mu(\mathbf{x})$ more aggressively by using 20 gradient ascent steps from each datapoint. This setting is a little different from our results in Table 1, where we find that generating adversarial samples $\mu(\mathbf{x})$ by using a single gradient step from each datapoint suffices to obtain good performance while also being compute-wise less intensive. We set $\beta = 0.0$ (Equation 4) in this experiment in order to test the affect of τ in isolation of other confounding factors. As shown in Figure 6, we vary the parameter τ from 0.5 to 0.001 and visualize the performance of solutions found by COMs as a function of the iteration of gradient ascent. Figure 6 demonstrates that increasing τ can improve the stability of solutions found by COMs, but if τ is large, solutions may converge too slowly. We empirically find an intermediate value of $\tau = 0.05$ is appropriate universally, and effective on a wide range of offline MBO tasks, and we utilize this setting universally to obtain results in Table 1.

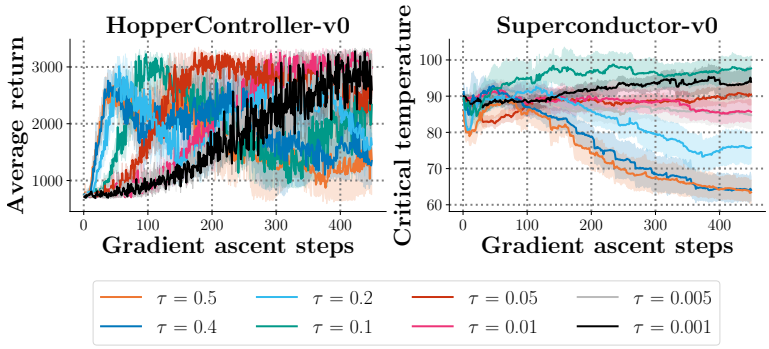


Figure 6. **Ablation of sensitivity and universality of τ.** In each of the two plots, we instantiate COMs on the HopperController-v0 and Superconductor-v0 tasks, and vary τ that controls the degree to which $\hat{f}_\theta$ conservatively estimates the ground truth. The x-axis denotes the number of gradient ascent steps taken to update the design $\mathbf{x}^*$ to maximize $\hat{f}_\theta$, and the y-axis indicates the resulting values of the ground truth function y, which remains unobserved by the COMs algorithm during training, and only serves as an ablative visualization. The results demonstrate that increasing τ results in improved stability, at the expense of slower convergence. A value of $\tau = 0.05$ is an appropriate *universal* choice of τ.

A.2. Optimization with Trust-Region Gradient Ascent

As per Equation 4, COMs updates performs gradient ascent on inputs $\mathbf{x}$, initialized from a particular $\mathbf{x}_0 \in \mathcal{D}$. This gradient ascent procedure utilizes a "trust-region" objective as restated below:

$$\forall\, t \in [T], \mathbf{x}_0 \in \mathcal{D}; \quad \mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla_{\mathbf{x}} \mathcal{L}_{\text{opt}}(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_t}$$
$$\text{where } \mathcal{L}_{\text{opt}}(\mathbf{x}) := \hat{f}^\star_\theta(\mathbf{x}) - \beta \hat{f}^\star_\theta(\mathbf{x} + \eta \nabla_{\mathbf{x}} \hat{f}^\star_\theta(\mathbf{x})). \tag{7}$$

The value of β here is technically a hyperparameter, and we find $\beta = 0.9$ to be an appropriate choice *universally* across all the tasks. One practical consideration of this style of update is that the term $\mathbf{x} + \eta \nabla_{\mathbf{x}} \hat{f}^\star_\theta(\mathbf{x})$ depends on the current solution particles $\mathbf{x}$, and thus the gradient $\nabla_{\mathbf{x}} \mathcal{L}_{\text{opt}}(\mathbf{x})$ contains a second-order term (i.e., Hessian of the function $\hat{f}_\theta$ with respect to $\mathbf{x}$). We empirically found that accounting for this second-order dependency is crucial and ignoring it leads to diminishing performance. Finally, a last consideration of this objective is how to tune the value of β. First note that in the formulation above, β always takes a value in a bounded interval $[0, 1]$: $\beta \in [0, 1]$. This is because any value of β outside this interval does not maximize the function $\hat{f}_\theta(\mathbf{x})$ and instead attempts to reduce its value in an overall sense. In practice, a coarse search in intervals of 0.1 was enough in our experiments to identify a *universal* value of β.

Hyperparameter	Value
Number of epochs to train $\hat{f}_\theta$	500
T (Number of gradient ascent steps using Equation 4)	450
α learning rate (used to optimize Equation 5 via dual gradient descent)	0.05
τ in Equation 5	0.05
β in Equation 4	0.9
Number of steps used to generate adversarial samples $\mu(\mathbf{x})$ in Equation 5	1
η in Equation 4	0.01

Table 2. **Hyperparameters for COMs.** All hyperparameters are kept constant across all tasks in COMs.

A.3. Implementation details

In addition to various considerations from Sections 3.3 and 3.4, one important implementation detail of COMs is to normalize the inputs ($\mathbf{x}$) and outputs (y values) for training the conservative model, $\hat{f}_\theta(\mathbf{x})$. Our motivation for using normalization was simple: Since the input and output ranges and modalities of various tasks we evaluated on in Table 1 is very different from each other, in order to be able to use a *uniform* set of hyperparameters for COMs, it is necessary to normalize both the inputs $\mathbf{x}$ and outputs y to the same range. Following standard normalization practices, we normalized $\mathbf{x}$ and y such that the resulting first and second moments match those of a unit Gaussian distribution. In practice, this means collecting all objective values from the training dataset into a vector $Y \in \mathbb{R}^{N \times 1}$, evaluating the sample mean $\hat{\mu} = \text{mean}(Y)$ and sample standard deviation $\hat{\sigma} = \text{std}(Y - \hat{\mu})$. A similar procedure is used for calculating the sample mean and sample standard deviation of $\mathbf{x}$. The objective values and inputs are then normalized by subtracting their sample mean and dividing by their sample standard deviation $y \leftarrow (y - \hat{\mu})/\hat{\sigma}$. This normalization allows COMs to use the uniform set of hyperparameters, which we mention explicitly, in Table 2.

A.4. Trust-Region Gradient Descent (Equation 4) and Equation 3

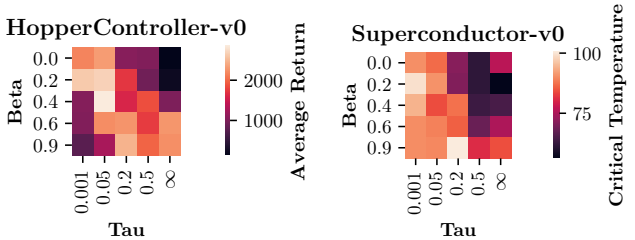
A.4.1. Does Equation 4 outrun the region defined by $\mu(\mathbf{x})$, as shown in Table 2?

In this section, we present an empirical analysis that shows that even when 450 steps of Equation 4 are used, the learned conservative model, $\hat{f}(\mathbf{x})$ satisfies the conservatism constraint satisfied by $\mu(\mathbf{x})$ during training, i.e., $\hat{f}(\mathbf{x}^*) - f(\mathbf{x}_0) \leq \tau$, where $\mathbf{x}^*$ denotes the point found by the trust-region optimizer and $\mathbf{x}_0$ is the initial point for Eqn. 4. For e.g., on HopperController, we find $\hat{f}(\mathbf{x}^*) - f(\mathbf{x}_0) = 1.18$ while $\tau = 2.0$ and on AntMorphology, $\hat{f}(\mathbf{x}^*) - f(\mathbf{x}_0) = 0.39$ while $\tau = 2.0$. Thus the trust-region optimizer "**does not outrun**" the possible $\mathbf{x}$ values for which the function $\hat{f}(\mathbf{x})$ satisfies the conservatism constraint.

A.4.2. Can COMs utilize Equation 4 for computing $\mu(\mathbf{x})$, thereby unifying the two procedures?

We now turn to answering the question if COMs can utilize Equation 4 used to compute $\mathbf{x}^*$, to also compute $\mu(\mathbf{x})$, thus unifying the two procedures. To empirically demonstrate that this is possible, we perform an experiment on two tasks and find that *COMs perform equally well* when the trust-region procedure in Equation 4 replaces $\mu(\mathbf{x})$ in Equation 3 (while training $\hat{f}^*$): HopperController: 2801.4 ± 282.0 vs 2864.5 ± 535.0 for base COMs; AntMorphology: 443.1 ± 8.9 vs 442.1 ± 8.3 for base COMs.

A.4.3. Ablation study of τ (Equation 5) vs β (Equation 4)



Finally, we present an ablation to understand the relative roles played by β and τ in ensuring conservatism in COMs. While these hyperparameters are used in two different scenarios: τ is used to train $\hat{f}(\mathbf{x})$, β is used to find $\mathbf{x}^*$, we note that they might appear to serve a similar functionality of preventing the optimizer from finding out-of-distribution designs by controlling the learned model that is being optimized (τ) or by preventing the optimizer from going too far away from the training data (β, trust-region). To understand the relative impacts of τ and β, we ran COMs with different values of τ and β and plot the resulting performance

on a heatmap for two domains (left: HopperController (HC), right: Superconductor (SC)). We find that the contribution of τ is crucial in COMs. For e.g., a specific value of τ can generally attain reasonably good performance agnostic of β in both cases (HopperController: 0.05, SuperConductor: $0.001/0.05$), whereas simply using Equation 4 with a standard objective model ($\tau = \infty$) which is trained only with supervised regression and not conservatively, doesn't attain the best results across β values, indicating that τ plays an important role in attaining good performance.

B. Proof of Theorem 1

In this section, we provide a proof for Theorem 1 and show that the conservative training by performing gradient descent on θ with respect to the objective in Equation 6 (restated below in a more convenient form as Equation 8) indeed obtains a conservative model of the actual objective function. Note that $\overline{\mathcal{D}}(\mathbf{x}'|\mathbf{x})$ denotes a smoothed Dirac-delta distribution centered at $\mathbf{x}$, which can be obtained by adding random noise to a given $\mathbf{x}$.

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

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

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

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

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

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

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

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

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

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

where the expression marked as $(*)$ denotes the $(k+1)$-th iterate of the function, under gradient descent on just the mean-squared error $(f(\mathbf{x}) - y)^2$ term, marked as $(\wedge)$ in Equation 8. Noting that the theorem statement denotes the term $(*)$ as $\tilde{f}_\theta^{k+1}(\mathbf{x}'')$, we obtain our first desired result. To obtain the first argument of the $\max$ in the theorem statement, note that if

the function $\hat{f}_\theta^{k+1}$ is $\widehat{L}$-Lipschitz, the value at $\mathbf{x}''$ cannot be smaller than $\hat{f}_\theta^{k+1}(\mathbf{x}) - \widehat{L}||\mathbf{x} - \mathbf{x}''||_2$, and hence the maximum over the two terms.

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

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

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

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

C. Network Details

In each of our experiments, we train a neural network $\hat{f}_\theta$ to approximate the ground truth score function of an offline MBO task, where θ represents the weights of the model. Distinct from prior methods based on generative models (Kumar & Levine, 2019; Brookes et al., 2019) we are able to utilize the same neural network architecture for representing the learned model, $\hat{f}_\theta(\mathbf{x})$ across all MBO tasks. This architecture is a three-layer neural network with two hidden layers of size 2048, followed by Leaky ReLU activation functions with a leak of 0.3. Each neural network $\hat{f}_\theta$ has an output layer that predicts a Gaussian distribution over potential objective values y by predicting both the mean $\mu_\theta(\mathbf{x})$ and log standard deviation $\log \sigma_\theta(\mathbf{x})$ of that distribution. For numerical stability, the log standard deviation is passed through a softplus function (rather than an exponential) to obtain standard deviation. The neural network $\hat{f}_\theta$ is trained to minimize negative log likelihood of observed data, using the default parameters of the Adam optimizer as discussed in Section 3.4.

D. Tasks and Data Collection

The tasks from this paper are taken from Design-Bench (Trabucco et al., 2021), a work-in-progress benchmark for evaluating offline model based optimization methods. We provide a copy of the paper in the supplementary material. 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?

D.1. Superconductor-v0

Superconductor-v0 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 the url `https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data#`. The original dataset consists of superconductors featurized into vectors $x_{\text{Superconductor}} \in \mathcal{R}^{81}$. One issue with the original dataset is that the largest value of a single dimension in the dataset is 22590.0, which appears to cause learning instability. We follow (Fannjiang & Listgarten, 2020) and normalize each dimension of the design-space to have zero mean and unit variance. However, we deviate from

the remaining pre-processing steps in (Fannjiang & Listgarten, 2020). In order to promote task realism, we directly use the superconductivity data, whereas (Fannjiang & Listgarten, 2020) re-samples by collecting iid unit gaussian samples and labelling them with the task oracle function. This causes the scores in the dataset to correspond exactly to the scores provided by the oracle. No other domain in design-bench re-samples nor re-labels static data, so we omit it here for consistency.

D.2. HopperController-v0

The HopperController task is one that we provide ourselves. The goal of this task is to design a set of weights for as neural network policy, in order to achieve high expected return when evaluating that policy. The data collected for HopperController was taken by training a three layer neural network policy with 64 hidden units and 5126 total weights on the Hopper-v2 MuJoCo task using Proximal Policy Optimization (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 $x_{\text{Hopper}} \in \mathcal{R}^{5126}$. The result is an optimization problem over neural network weights. After collecting these weights, we perform no additional pre-processing steps. In order to collect scores we perform a single rollout for each x using the Hopper-v2 MuJoCo environment. The horizon length for training and evaluation is limited to 1000 simulation time steps.

D.3. AntMorphology-v0 & DKittyMorphology-v0

Both morphology tasks are collected by us with the same methodology. The goal of these tasks is to design the morphology of a quadrupedal robot—an ant or a D'Kitty—such that the agent is able to crawl quickly in a particular direction. In order to collect data for this environment, we create variants of the MuJoCo Ant and the ROBEL D'Kitty agents that have parametric morphologies. The goal is to determine a mapping from the morphology of the agent to the average return that an agent trained for a particular intended morphology achieves. We implement this by pre-training a neural network policy using SAC (Haarnoja et al., 2018). For both the Ant and the D'Kitty, we train agents for up to three million environments steps, and a maximum episode length of 1000, with all other settings as default. These agents are pre-trained for a fixed gold-standard morphology—the default morphology of the Ant and D'Kitty respectively. Each morphology task consists of samples obtained by adding Gaussian noise with standard deviation 0.02 for Ant and 0.01 for DKitty times the design-space range to the gold-standard morphology. We label each sampled morphology by averaging the return of 16 rollouts of length 100 of an agent with that morphology.

E. Oracle Functions

We detail oracle functions for evaluating ground truth objective values for each of the tasks in Table 1.

E.1. Superconductor-v0

The Superconductor-v0 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 Random Forest Regressor provided in scikit-learn. Similar to the setup for the previous two tasks, this oracle is trained on the entire static dataset, and the task is instantiated with a split percentile. Samples scoring at most in the 80th percentile are observed by an MBO algorithm, which allows for sampling *unobserved* points that score in the unobserved top 20 percent.

E.2. HopperController-v0

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

over 1000 steps.

E.3. AntMorphology-v0 & DKittyMorphology-v0

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