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

The TF Bind 8 task is a derivative of the transcription factor binding activity survey performed by [5], where the binding activity scores of every possible length eight DNA sequence was measured with a variety of human transcription factors. We filter the dataset by selecting a particular transcription factor `SIX6_REF_R1`, and defining an optimization problem where the goal is to synthesize a length 8 DNA sequence with high binding activity with human transcription factor `SIX6_REF_R1`. This particular transcription factor for TF Bind 8 was recently used for optimization in [2, 3]. TF Bind 8 is a fully characterized dataset containing 65792 samples, representing every possible length 8 combination of nucleotides $\mathbf{x}_{\text{TFBind8}} \in \{0, 1\}^{8\times 4}$. The training set given to offline MBO algorithms is restricted to the bottom 50%, which results in a visible training set of 32898 samples.

A.2 GFP

The GFP task provided is a derivative of the GFP dataset [30]. The dataset we use in practice is that provided by [7] at the url `https://github.com/dhbrookes/CbAS/tree/master/data`. We process the dataset such that a single training example consists of a protein represented as a tensor $\mathbf{x}_{\text{GFP}} \in \{0, 1\}^{237\times 20}$. This tensor is a sequence of 237 one-hot vectors corresponding to which amino acid is present in that location in the protein. We use the dataset format of [7] 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 [7].

A.3 UTR

The UTR task is derived from work by Sample et al. [29] 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. [3], where the goal is to design a length 50 DNA sequence to maximize expression level. We follow the methodology set by Sample et al. [29] to sort all length 50 DNA sequences in the unprocessed UTR dataset by total reads, and then select the top 280,000 DNA sequences with the most total reads. The result is a dataset containing 280,000 samples of length 50 DNA sequences $\mathbf{x}_{\text{UTR}} \in \{0, 1\}^{50\times 4}$ and corresponding ribosome loads. When training offline MBO algorithms, we subsequently eliminate the top 50% of sequences ranked by their ribosome load, resulting in a visible dataset with only 140,000 samples.

A.4 ChEMBL

The ChEMBL task is a derivative of a much larger dataset that is derived from ChEMBL [13], a large database of chemicals and their properties. The data, similar to GFP, was originally collected by performing physical experiments on a large number of molecules, and measuring their activity with a target assay. We have processed the 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 = CHEMBL1964047` and select the standard type of `GI50` as the measurement to maximize with offline model-based optimization. The resulting dataset has 40516 samples in total. 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. The resulting processed samples have a maximum sequence length of 425 and the vocabulary has 591 elements, $\mathbf{x}_{\text{ChEMBL}} \in \{0,1\}^{425\times 591}$. The final step is to remove the top 47% of molecules sorted by their `GI50` value. This step removes a mode of the dataset that we found to hurt the performance of learned oracle models trained on this processed dataset.

A.5 Superconductor

The Superconductor task is inspired by recent work [10] 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 [16], 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 [10] is that the numerical representation of the superconducting materials did not lend itself to recovering a physically realizable material that could be synthesized in a lab after performing model-based optimization. In order to create an *invertible* input specification, we deviate from prior work and encode superconductors as vectors whose components represent the number of atoms of specific chemical elements present in the superconducting material—a serialization of the chemical formula of each superconductor. The result is a real-valued design space with 86 components $\mathbf{x}_{\text{Superconductor}} \in \mathcal{R}^{86}$. The full dataset used to learn approximate oracles for evaluating MBO methods has 21263 samples, but we restrict this number to 17010 (the 80th percentile) for the training set of offline MBO methods to increase difficulty.

A.6 Hopper Controller

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 [31]. Specifically, we use the default parameters for PPO provided in stable baselines [18]. The dataset we provide with this benchmark has 3200 unique weights. In order to collect this many, we run 32 experimental trials of PPO, where we train for one million steps, and save the weights of the policy every 10,000 environment steps. The policy weights are represented originally as a list of tensors. We first traverse this list and flatten each of the tensors, and we then concatenate each of these flattened tensors into a single training example $\mathbf{x}_{\text{Hopper}} \in \mathcal{R}^{5126}$. The result is an optimization problem over neural network weights. After collecting these weights, we perform no additional pre-processing steps. In order to collect scores we perform a single rollout for each x using the Hopper-v2 MuJoCo environment. The horizon length for training and evaluation is limited to 1000 simulation time steps, which is standard practice for this MuJoCo environment.

A.7 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 a pre-trained morphology conditioned agent. We implement this by pre-training a morphology conditioned neural network policy using SAC [15]. For both the Ant and the D'Kitty, we train the agents for more than ten million environment steps, and a maximum episode length of 200, with all other settings as default. These agents are pre-trained on Gaussian distributions of morphologies. The Gaussian distributions are obtained by adding Gaussian noise with standard deviation 0.03 for Ant and 0.01 for D'Kitty the design-space range to the default morphologies. After obtaining trained morphology-conditioned policies, we create a dataset of morphologies for model-based optimization by sampling initialization points randomly, and then using CMA-ES to optimize for morphologies that attain high reward using the pretrained morphology-conditioned policy. To obtain initialization points, we add Gaussian random noise to the default morphology for the Ant with standard deviation 0.075 and D'Kitty with standard deviation 0.1, and then apply CMA-ES with standard deviation 0.02. We ran CMA-ES for 250 iterations and then restart, until a minimum of 25000 morphologies were collected, resulting in a

final dataset size of 25009 for both the Ant and D'Kitty. The design space for Ant Morphologies is $\mathbf{x}_{\text{Ant}} \in \mathcal{R}^{60}$, whereas for D'Kitty morphologies is $\mathbf{x}_{\text{D'Kitty}} \in \mathcal{R}^{56}$. We subsample the dataset to its 40th percentile when training offline MBO algorithms, resulting in 10004 samples.

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

TF Bind 8 is a fully characterized discrete offline MBO task, which means that all possible designs have been evaluated [5] and are contained in the full hidden TF Bind 8 dataset. The oracle for TF Bind 8 is therefore implemented as a lookup table that returns the score corresponding to a particular length 8 DNA sequence from the dataset. By restricting the size of the training set visible to an offline MBO algorithm, it is possible for the algorithm to propose a design that achieves a higher score than any other DNA sequence visible to the offline MBO algorithm during training.

B.2 GFP

GFP uses the oracle function derived from Rao et al. [27]. 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.

B.3 UTR

UTR uses a Transformer as the oracle function, which differs from the CNN that was originally used by [3]. 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. [29]. 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 Transformer has a spearman's rank correlation coefficient of 0.6424 with a held-out validation set.

B.4 ChEMBL

We tested several models as candidate oracle functions for ChEMBL [13], including Gaussian Process, Random Forest, CNN, and Transformer regression models. We ultimately found the CNN to result in the highest validation performance, achieving a spearman's rank correlation coefficient of 0.3208 with a held-out validation set. These models were trained on the entire hidden ChEMBL dataset encoded into SMILES and tokenized. While this rank correlation is low in comparison to the previous tasks, such illustrates the challenge learning of predicting molecule properties directly from SMILES, and is a welcome avenue for future work in the design an training of such models. Our CNN consists of four residual blocks each with two convolution layers with kernel size 3 and 64 filters.

B.5 Superconductor

The Superconductor oracle function is also a random forest regression model. The model we use it the model described by [16]. We borrow the hyperparameters described by them, and we use the RandomForestRegressor provided in scikit-learn. Similar to the setup for the previous set of tasks, this oracle is trained on the entire hidden dataset of superconductors. The random forest has a spearman's rank correlation coefficient with a held-out validation set of 0.9155.

B.6 HopperController

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

B.7 Ant & D'Kitty Morphology

The final two tasks in design-bench use an exact oracle function, using the MuJoCo simulator. For both morphology tasks, the simulator performs a rollout and returns the sum of rewards at every timestep in that rollout. Each task is accompanied by a pre-trained morphology-conditioned policy. To perform evaluation, a morphology is passed to the Ant or D'Kitty MuJoCo environments respectively, and a dynamic-morphology agent is initialized inside these environments. These simulations can be time consuming to run, and so we limit the rollout length to 100 steps. The morphology conditioned policies were trained using the reinforcement learning algorithm SAC for 10 million steps for each task, and are ReLU networks with two hidden layers of size 64.

C Experimental Details

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

C.1 Score 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.

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

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.

	GFP	TF Bind 8	UTR	ChEMBL
Autofocus.-CbAS	0.848 ± 0.007	0.419 ± 0.007	0.517 ± 0.010	0.216 ± 0.000
CbAS	0.852 ± 0.004	0.428 ± 0.010	0.515 ± 0.014	0.243 ± 0.006
BO-qEI	0.246 ± 0.341	0.439 ± 0.000	0.574 ± 0.000	0.298 ± 0.034
CMA-ES	0.047 ± 0.000	0.537 ± 0.014	0.385 ± 0.019	0.316 ± 0.031
Gradient Ascent	0.838 ± 0.004	0.609 ± 0.019	0.489 ± 0.009	0.300 ± 0.031
Grad. Min.	0.837 ± 0.001	0.645 ± 0.030	0.514 ± 0.007	0.306 ± 0.023
Grad. Mean	0.838 ± 0.002	0.616 ± 0.023	0.508 ± 0.005	0.326 ± 0.016
MINs	0.820 ± 0.018	0.421 ± 0.015	0.560 ± 0.007	0.347 ± 0.001
REINFORCE	0.844 ± 0.003	0.462 ± 0.021	0.530 ± 0.008	0.246 ± 0.004
	Superconductor	Ant Morphology	D'Kitty Morphology	Hopper Controller
Autofocus.-CbAS	0.131 ± 0.010	0.362 ± 0.015	0.736 ± 0.025	0.019 ± 0.008
CbAS	0.111 ± 0.017	0.382 ± 0.016	0.753 ± 0.008	0.015 ± 0.002
BO-qEI	0.300 ± 0.015	0.566 ± 0.000	0.883 ± 0.000	0.343 ± 0.010
CMA-ES	0.379 ± 0.003	-0.050 ± 0.004	0.684 ± 0.016	-0.033 ± 0.005
Gradient Ascent	0.476 ± 0.022	0.130 ± 0.018	0.509 ± 0.200	0.092 ± 0.084
Grad. Min	0.471 ± 0.016	0.182 ± 0.008	0.746 ± 0.034	0.222 ± 0.065
Grad. Mean	0.469 ± 0.022	0.184 ± 0.010	0.748 ± 0.024	0.243 ± 0.064
MINs	0.336 ± 0.016	0.619 ± 0.041	0.887 ± 0.004	0.352 ± 0.058
REINFORCE	0.463 ± 0.016	0.134 ± 0.033	0.356 ± 0.131	-0.064 ± 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
.

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.

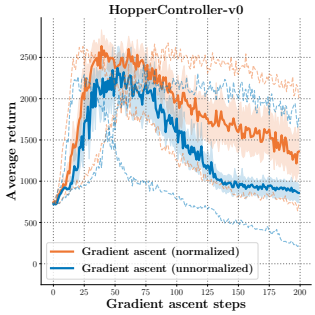
	GFP	TF Bind 8	UTR	ChEMBL
Autofocus CbAS	3.739 ± 0.001	0.910 ± 0.044	7.794 ± 0.070	42467.285 ± 0.000
CbAS	3.738 ± 0.001	0.927 ± 0.051	7.800 ± 0.023	46681.988 ± 4987.456
BO-qEI	2.005 ± 1.000	0.798 ± 0.083	7.903 ± 0.000	30069.684 ± 3187.300
CMA-ES	1.438 ± 0.005	0.953 ± 0.022	7.993 ± 0.052	31607.031 ± 1578.222
Gradient Ascent	3.737 ± 0.002	0.977 ± 0.025	7.669 ± 0.113	32514.541 ± 2612.903
Grad. Min	3.738 ± 0.000	0.984 ± 0.012	7.758 ± 0.081	32617.006 ± 370.390
Grad. Mean	3.738 ± 0.000	0.986 ± 0.012	7.764 ± 0.058	33715.059 ± 1136.034
MINs	3.741 ± 0.002	0.905 ± 0.052	7.787 ± 0.053	42732.578 ± 5126.862
REINFORCE	3.739 ± 0.001	0.948 ± 0.028	7.747 ± 0.059	41448.012 ± 3220.380
	Superconductor	Ant Morphology	D'Kitty Morphology	Hopper Controller
Autofocus CbAS	77.910 ± 8.361	474.888 ± 44.424	226.156 ± 7.043	261.820 ± 6.366
CbAS	93.078 ± 12.695	469.499 ± 30.570	209.412 ± 9.593	267.623 ± 15.833
BO-qEI	74.322 ± 6.347	413.084 ± 0.000	213.816 ± 0.000	788.990 ± 149.878
CMA-ES	86.072 ± 4.508	799.394 ± 715.702	4.290 ± 1.505	857.178 ± 273.980
Grad.	95.789 ± 4.436	-100.265 ± 22.118	187.206 ± 27.274	1406.413 ± 613.631
Grad. Min	93.590 ± 1.719	80.853 ± 62.308	205.639 ± 13.427	1860.192 ± 750.026
Grad. Mean	92.265 ± 3.206	48.064 ± 78.555	209.355 ± 13.928	2108.004 ± 578.350
MINs	86.702 ± 4.171	505.515 ± 34.934	273.479 ± 14.184	628.436 ± 210.953
REINFORCE	88.996 ± 2.389	-127.440 ± 30.831	-194.540 ± 238.857	62.183 ± 84.937

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.

C.4 Computation Resources

The amount of computation resources required to produce the experiments in this paper is relatively modest. 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 Normalization Of Inputs and Outputs For Gradient Ascent Baseline



An important component for the the good performance of gradient ascent baseline is the normalization of design space. We found that the identical gradient-ascent baseline performed a factor 1.4x worse on HopperController, when optimizing in the space of unnormalized designs and objective values, as seen in Figure D. 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 normalize only the scores to have unit Gaussian statistics. This is a necessary part of the optimization workflow because scores vary by several orders of magnitude in the dataset, for example, 0.864 for GFP similar to other methods and as high as 1.586 for HopperController. The specific normalization equation for continuous-valued designs is given below.

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

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

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

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.

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

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.

E Hyperparameter Selection Workflow

Hyperparameter tuning under a restricted computational budget is emerging as an import research domain in optimization [34, 9, 19]. 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: https://github.com/brandontrabucco/design-baselines.

We now present specific guidelines for hyperparameter selection (i.e. ***workflow***) for some of the methods evaluated in our benchmark. These principles are general principles that can be used to tune the hyperparameters of these methods on a new task in a completely 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.

E.1 Strategy For Autofocused CbAS

The main tunable components of Autofocused methods [10] 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.

E.2 Strategy For CbAS

The main tunable components of CbAS methods [7] 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 E.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 E.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 E.1.

E.3 Strategy For MINs

The main tunable components of MINs [22] 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 E.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 $I(\mathbf{x}; \mathbf{z}) > c$.

E.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 E.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 model-selection criterion to select this parameter is future work.

E.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 E.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. If the stability of the gradient estimator is suffering, due to high-variance updates, a density ratio threshold like in PPO [31] can be applied, and a baseline can be subtracted.

E.6 Strategy For Bayesian Optimization

The main tunable components of Bayesian Optimization MBO methods [4] 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 E.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.

E.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 E.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 Fidelity of Expert Model Oracle Functions

In this section we present additional studies on the quality of expert model oracle functions we use in our benchmark, as high fidelity oracles are used for evaluation in this benchmarks on tasks where real evaluation using the actual ground truth objective requires running real experiments. We first analyze the train-test discrepancy of the oracle, where we use the oracle's predicted y values instead of the dataset values for the MBO algorithm. We measure the correlation of the performance ranks of all baseline algorithms in our benchmark, and we also measure of the change of rank of different algorithms. High correlation and low rank shift indicate that the use of oracle predicted values does not affect the relative performance order of different offline MBO algorithms, suggesting the benchmark is robust. We present the results in Table 5. For the majority of MBO tasks where an exact oracle is not available, the impact of the train-test discrepancy is minimal. For UTR, the task most sensitive to the train-test discrepancy, an option in the Design-Bench code-base allows for the training set to be relabelled with the predictions of the oracle model and the train-test discrepancy to be removed.

	GFP	UTR	Superconductor
Rank Correlation	0.895405	0.116667	0.666667
Max Rank Shift	2.000000	7.000000	4.000000
Avg Rank Shift	0.888889	2.666667	1.333333

Table 5: Spearman's rank correlation coefficient of MBO algorithms when (1) using raw measurements in the training set and a learned oracle during evaluation versus (2) using predictions from a learned oracle during training and during evaluation.

In addition to the train-test discrepancy, we also measure the oracle agreement between different expert oracle models on the same dataset. To quantitatively analyze the agreement on different oracles, we use them to evaluate all offline MBO algorithms in this dataset, and measure the changes ranks of offline MBO algorithms between oracles. We present the results in Figure 4. The results suggest that high quality expert oracle models do agree with each other on the solutions found by offline MBO algorithms, indicating that the expert oracles in our benchmark is robust.

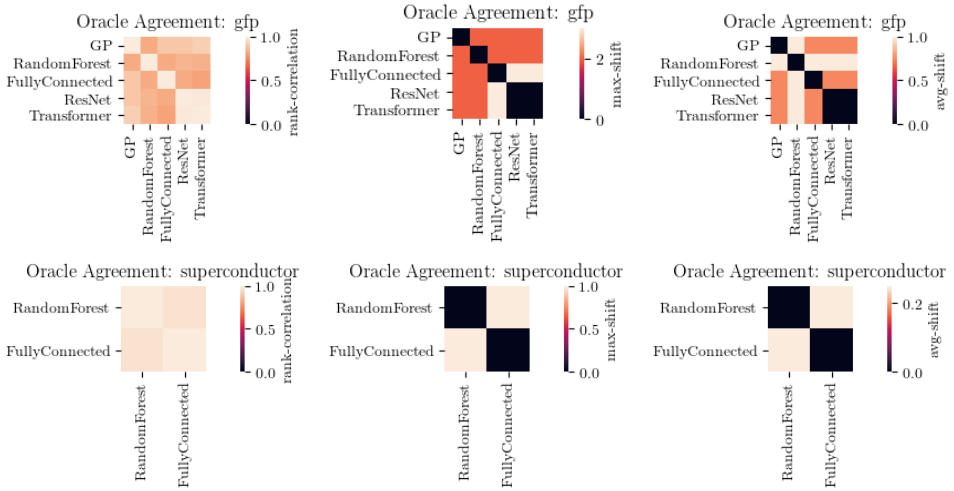


Figure 4: **Agreements of the ranks of offline MBO methods using different oracles on the same dataset**. We evaluate all offline MBO algorithms benchmarked in this paper on the same dataset using different oracles, and compare their performance ranking under different oracles. The rank correlation measures Spearman's rank correlation coefficient between the order of the algorithms, and the mean and max rank shift shows the average and the max rank change of the same algorithm between different oracles. We can see that the rank correlations are high and rank shifts are low, suggesting that different oracles tend to agree on the relative order of the performance of offline MBO algorithms. This result indicates that our expert oracle models are robust.