# Data-Driven Model-Based Optimization via Invariant Representation Learning

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

We study the problem of data-driven model-based optimization (MBO), where the goal is to find the optimal design, provided access to only a static dataset, with no active data collection. The central challenge in data-driven MBO is distributional shift, where the optimizer is fooled into producing out-of-distribution (OOD) designs that erroneously appear promising under a model trained on the provided data. To address this issue, we formulate MBO as *domain adaptation*, where the goal is to make accurate predictions for the value of designs during optimization ("target domain"), when training only on the dataset ("source domain"). This perspective leads to invariant objective models (IOM), our approach for addressing distributional shift by enforcing invariance between the learned representations of the training dataset and optimized designs. In IOM, if the optimized designs are too different from the training dataset, the representation will be forced to lose much of the information that distinguishes good designs from bad ones, making all choices seem mediocre. Critically, when the optimizer is aware of this representational tradeoff, it should choose not to stray too far from the training distribution, leading to a natural trade-off between distributional shift and learning performance.

## 1 Introduction

Many real-world applications of machine learning involve learning how to make better designs from data. This is formulated as data-driven model-based optimization (also known as black-box optimization [2]), in which an algorithm is provided with data from previous experiments consisting of $(\mathbf{x}, y)$ tuples, where $\mathbf{x}$ represents a design (e.g., a molecule [14] or robot controller [5]) and $y$ represents its utility (e.g., certain property of the molecule or the effectiveness of a controller). The goal is to generate the best possible design, typically one that is better than the best one in the dataset. All of these problem domains are united by a common challenge: in order to improve, the designs learned by the algorithm must differ from the distribution of the designs in the training data, inducing *distributional shift* [26], which must be carefully handled.

Typical methods for solving such data-driven model-based optimization problems try to learn some form of a proxy model $\hat{f}(\cdot)$ that maps a design $\mathbf{x}$ to its objective value $y$ via supervised learning on the static dataset, and then optimize the design $\mathbf{x}$ against this learned model. However, overestimation errors in the learned model will erroneously drive the optimizer towards low-value designs that "fool" the proxy into producing a high values [26]. To handle this, prior works have proposed to regularize the learned model to be conservative by penalizing its values on unseen designs [46, 51, 20], or explicitly constraining the optimizer (e.g., by using density estimation) [8, 26].

In this paper, we take a different perspective for the distributional shift challenges in the model-based optimization, and formulate it as domain adaptation, where the "target" domain consists of the distribution of designs that are chosen by the optimizer, and the source domain corresponds to the training set. Framing the problem in this way, we derive a class of methods that regulate distributional

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

shift at the *representation level*, and admit effective hyperparameter tuning strategies. The intuition behind our approach is that, if the internal representations of the learned model are made *invariant* to the differences between the designs seen in the training data vs. the optimized distribution, then excessive distributional shift is naturally discouraged because excessively invariance would lead the out-of-distribution points to attain mediocre values on average. That is, as the optimized decisions deviate more from the training data, the requirement to maintain invariance will cause the model's representation to lose information, and will hence force the predicted value to become closer to the average $y$ value in the dataset. This will force the optimizer to stay within the distribution of the dataset, since out-of-distribution points will only be mediocre under the learned model. Of course, the representation cannot be perfectly invariant in general, because the optimized points are different from the training data, and hence the method must strike a balance. In practice, one can instantiate this idea using an adversarial training procedure.

Our main contribution is an offline, data-driven model-based optimization method based on a novel connection with domain adaptation. Our approach, IOM, optimizes the design against a learned model that admits invariant representations between optimized designs and the dataset. This enables controlling distributional shift and also provides a way to derive effective offline hyperparameter tuning strategies that we show actually work well in practice. IOM can be implemented simply by combining any supervised regression procedure with a regularizer that enforces invariance. The regularizer is implemented using any discrepancy measure between two distributions, and we instantiate IOM using the $\chi^2$-discrepancy measure via the discriminator of a least-squares generative adversarial network. This procedure trains a discriminator to discriminate between the *representations* on the training dataset and optimized designs, whereas the learned representations are trained to be as invariant as possible to fool the discriminator. We theoretically derive IOM from our domain adaptation formulation of MBO and we also use this formulation to derive tuning strategies for several common hyperparameters. Empirically, we evaluate IOM on several tasks from Design-Bench [45] and find that it outperforms the best prior methods, and additionally, admits appealing offline tuning strategies unlike the prior methods.

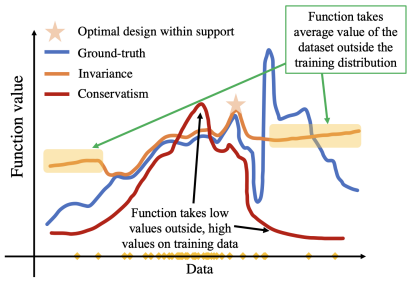


Figure 1: **Illustration showing the intuition behind IOM** on a 1D task. While conservatism (COMs) [46] pushes down the learned values on designs not observed in the dataset and pushes up the learned values on designs in the dataset, training with invariance induces the out-of-distribution designs to attain objective values close to the average objective value in the training dataset. This discourages the optimizer from finding out-of-distribution designs.

## 2 Related Work

Model-based optimization (MBO) or black-box optimization [8, 3, 43, 42, 35, 51] has been studied widely in prior works. Among these works, the central challenge is in handling the distributional shift that arises between the distribution of designs seen in the dataset and those taken by the optimizer [8, 11, 26]. For methods that learn a model, this typically manifests itself as *model exploitation*, where the optimizer can find out-of-distribution designs for which the model erroneously predicts good values, analogously to adversarial examples. This distributional shift is distinct from what is studied in prior works that tackle misspecification in GPs and bandits [36, 37, 24] or methods that tackle robustness across contexts [25] in the online setting.

To address distribution shift, several recent works have proposed to be conservative. This is majorly the case in other decision-making problems offline reinforcement learning (RL) [32] and contextual bandits [38], where instead of optimizing for a design, we must find the optimal action at each state. In offline MBO, prior works have tried to be conservative in two different ways: either by constraining the optimized designs to lie close to the training designs [8, 11, 26, 39, 27] using generative models [23, 15], or by learning a conservative learned model [46, 29, 28] or calibrated [12, 51]. While the latter class of methods has worked well in practice [45, 29], these methods can sometimes be overly conservative and are quite sensitive to hyperparameters. Sensitivity to hyperparameters is acceptable as long as one can attain good performance using reasonable offline tuning strategies, without accessing the groundtruth function, but we find in our experiments (Section 6.2) this is not easy for conservative methods. On the contrary, IOM not only does not require a generative model over the design space, which can be intractable with high-dimensions (e.g., 5000-d designs in HopperController), attains better performance, and provides a convenient tuning rule.

In this work we formulate data-driven model-based optimization as domain adaptation [4, 13], and derive algorithms for model-based optimization from this perspective. This perspective has been used to devise decision-making methods for treatment effect estimation [21, 22, 50] and off-policy evaluation in RL [33, 31]. While these prior works do use an invariance regularizer, they primarily focus on obtaining more accurate estimations, whereas our goal is to utilize invariance to combat distributional shift during optimization. Thus, our usage of invariant representations is tailored towards the optimizer, instead of aiming for accurate predictions everywhere like these prior works, which is impossible in general. Furthermore, our analysis uses invariant representations to derive procedures for tuning and early stopping, while these prior works do not explore these perspectives.

## 3 Preliminaries

The goal in offline model-based optimization (MBO) [45] is to produce designs that maximize some objective function $f(\mathbf{x})$, by only utilizing a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ of designs $\mathbf{x}_i$ and their corresponding objective values $y_i$. No active queries to the groundtruth function are allowed. Typically offline MBO is formulated as finding the optimal design $\mathbf{x}^\star$. However, we will use a slightly more general formulation, where the goal is to optimize a *distribution* over designs $\mu(\mathbf{x})$. We will use this formulation largely for convenience of analysis, where the classic case is recovered when $\mu(\mathbf{x})$ is a Dirac delta function, though we also note that optimizing distributions over designs is often studied in settings such as synthetic biology, where the goal is to explicitly diversify the designs [6]. Thus, our goal is to find a distribution over designs $\mu_{\text{OPT}}(\mathbf{x})$ using only the static dataset $\mathcal{D}$, such that it maximizes the *expected value* of $f(\mathbf{x})$ (the expected value under distribution $\mu$ is referred to as $J(\mu)$). This can be formalized as follows (Alg is an algorithm):

$$\mu_{\text{OPT}} := \arg\max_{\text{Alg}} \mathbb{E}_{\mathbf{x}\sim\mu(\mathbf{x})}[f(\mathbf{x})] := J(\mu) \quad \text{s.t.} \;\; \mu = \text{Alg}(\mathcal{D}). \tag{1}$$

The methods we consider in this paper fit a learned model $f_\theta(\mathbf{x})$ to $(\mathbf{x}_i, y_i)$ pairs in the dataset via a standard supervised regression objective, with an additional regularizer. We will assume that $f_\theta$ is composed of two components: a representation $\phi \in \mathbb{R}^d$ that maps a given $\mathbf{x}$ into a $d$-dimensional vector, and a forward model, $f_\theta$ that converts the representational output into a scalar objective value, i.e., $f_\phi(\mathbf{x}) := f_\theta(\phi(\mathbf{x}))$. We will use the notation $f_\theta$ and $f_\phi$ interchangeably. Given a learned model $f_\theta(\phi(\mathbf{x}))$, one approach to obtain the optimized distribution $\mu_{\text{OPT}}$ is by optimizing the expected value $J_\theta(\mu) := \mathbb{E}_{\mathbf{x}\sim\mu}[f_\theta(\phi(\mathbf{x}))]$ under the learned model. This distribution $\mu_{\text{OPT}}(\mathbf{x})$ can be instantiated in many ways: while some prior work [26, 8] represents $\mu_{\text{OPT}}$ via a parametric density model over $\mathcal{X}$ and optimizes this density model to obtain $\mu_{\text{OPT}}$, other prior work [46] uses a non-parametric representation for $\mu_{\text{OPT}}$ using a set of optimized "design particles". In this paper, we represent $\mu_{\text{OPT}}$ using a set of design particles, each of which is obtained by optimizing the learned function $f_\theta(\phi(\mathbf{x}))$ with respect to the input $\mathbf{x}$ via grad. ascent, starting from i.i.d. samples from the dataset. Formally:

$$\mathbf{x}_{k+1}^i \leftarrow \mathbf{x}_k^i + \eta \nabla_{\mathbf{x}} f_\theta(\phi(\mathbf{x}))|_{\mathbf{x}=\mathbf{x}_k^i}, \;\; \text{for} \;\; k \in [1, T], \quad \mathbf{x}_0^i \sim \mathcal{D} \tag{2}$$

The optimized distribution $\mu_{\text{OPT}}$ is then given by $\mu_{\text{OPT}} = \delta\{\mathbf{x}_T^1, \mathbf{x}_T^2, \cdots\}$.

**The challenge of distributional shift in offline MBO [26, 9, 11, 46].** When the learned model $f_\theta(\phi(\mathbf{x}))$ is trained via supervised regression on the offline dataset $\mathcal{D}$: $\arg\min_\theta \frac{1}{n}\sum_{i=1}^n (f_\theta(\phi(\mathbf{x}_i)) - y_i)^2$, the ERM principle states that $f_\theta(\phi(\mathbf{x}))$ will reasonably approximate $f(\mathbf{x})$ in expectation only under the distribution of the training dataset $\mathcal{D}$, i.e., $\mu_{\text{data}}$, however $f_\theta(\phi(\mathbf{x}_i)) \neq y_i$ in general on points not in the training distribution. Then, when optimizing the learned function, *overestimation errors* in the value of $f_\theta(\phi(\mathbf{x}))$ will inevitably lead the optimizer to converge to a distribution $\mu_{\text{OPT}}$ over designs that have a high value under the learned function (i.e., high $J_\theta(\mu_{\text{OPT}})$) but not a high ground truth value $J(\mu_{\text{OPT}})$. In general, this distribution $\mu_{\text{OPT}}$ will be different from $\mu_{\text{data}}$, since the learned function is likely to be more correct in expectation under $\mu_{\text{data}}$.

## 4 Data-Driven Offline Model-based Optimization as Domain Adaptation

In this section, we will formulate data-driven offline MBO as domain adaptation, which will allow us to derive our method, invariant objective models (IOM). Instead of attempting to simply constrain the optimizer to directly avoid distributional shift, our approach modifies the representations learned by the model $\hat{f}_\theta$ by adding an invariance regularizer during training such that there is minimal distribution shift under the learned invariant representation during optimization.

If we can ensure that the learned model $f_\theta$ is accurate on the distribution found by the optimizer, $\mu_{\text{OPT}}$, then we can ensure that we will obtain good designs. Therefore, to handle the distributional shift,

we can treat $\mu_{\text{data}}$ as the "source domain" and the optimized distribution $\mu_{\text{OPT}}$ as the "target domain." Just as the goal in domain adaptation is to train on the source domain and make accurate predictions on the target domain, we will aim to train $f_\theta(\mathbf{x})$ on $\mu$ and with the goal of making more accurate predictions under $\mu_{\text{OPT}}$. Of course, learning to make accurate predictions on *any* unseen distribution is impossible for any domain adaptation method [55], but in the case of MBO, the choice of the target distribution $\mu_{\text{OPT}}$ is also made by the algorithm itself, such that invariance can be enforced *both* by changing the representation and changing the target distribution $\mu_{\text{OPT}}$.

### 4.1 Invariant Objective Models (IOM)

IOM controls distributional shift between the optimized distribution $\mu_{\text{OPT}}$ and the training distribution $\mu_{\text{data}}$ at the representational level, by regularizing the distribution of the features $\phi(\mathbf{x})$ under $\mu_{\text{OPT}}$ to match that under $\mu_{\text{data}}$. We will first present the intuition behind the method, and then formalize it by setting up a bi-level optimization problem that we will also analyze theoretically.

**Intuition:** We supplement this intuition with an illustration shown in Figure 10. As the optimizer deviates from the training distribution $\mu$, the invariance regularizer forces the distribution of optimized designs to resemble the distribution over representations $\phi(\mathbf{x})$ under $\mu_{\text{data}}$, i.e., $\mathbb{P}_{\mu_{\text{OPT}}}(\phi(\mathbf{x})) \approx \mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x}))$. When $\mu_{\text{OPT}}$ is far from the training distribution (i.e., if $D_{\text{KL}}(\mu_{\text{OPT}}, \mu_{\text{data}})$ is very high), it is easier to enforce invariance as it does not conflict with fitting $f_\theta$ on the training distribution. This representational invariance in turn causes the expected value of the learned function $f_\theta$ under $\mu_{\text{OPT}}$ to appear (roughly) equal to the average objective value in the training dataset, i.e., $\mathbb{E}_{\mathbf{x}\sim\mu_{\text{OPT}}}[f_\theta(\phi(\mathbf{x}))] \approx \mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}}[f_\theta(\phi(\mathbf{x}))]$, if $\mu_{\text{OPT}}$ is too far away from the training distribution. As a result, the expected value of the learned model $f_\theta(\phi(\mathbf{x}))$ under $\mu_{\text{OPT}}$, $J_\theta(\mu_{\text{OPT}})$, will be worse than the value of the best design within the training distribution (since the best design input sampled from $\mu_{\text{data}}$ will attain a higher objective value than the average value under $\mu_{\text{data}}$). This would act as a guard to prevent the optimizer diverging too far away from the dataset, as out-of-distribution designs no longer appear promising. Iteratively regularizing $\phi$ to enforce invariance between $\mu_{\text{OPT}}$ and $\mu$ until convergence will restrict the optimizer from selecting erroneously overestimated out-of-distribution inputs. We will now present a formalization of this intuition in the form of a bi-level optimization problem that aims to maximize a lower-bound on the expected objective value attained under $\mu_{\text{OPT}}$, and derive the training objectives for IOM using this bi-level optimization formulation. However, do note that, our practical method does not aim to solve a bi-level optimization exactly.

**Bi-level optimization for invariant objective models (IOM).** To enforce invariance at a representational level, we utilize an invariance regularizer. Denoted as $\text{disc}_{\mathcal{H}}(p, q)$, our invariance regularizer is a discrepancy measure between two probability distributions $p$ and $q$, w.r.t. a certain hypothesis class $\mathcal{H}$. For example, if $\mathcal{H}$ is the class of all 1-Lipschitz functions, $\text{disc}_{\mathcal{H}}$ is the 1-Wasserstein distance; if $\mathcal{H}$ is the class of linear functions, $\text{disc}_{\mathcal{H}}$ is the maximum mean discrepancy (MMD) [16] distance. In practice, we implement this via a generative adversarial network as we discuss in the next subsection. In addition to training $f_\theta$ via standard supervised regression on the labels in the training data, IOM trains the representation $\phi(\mathbf{x})$ and the model $\widehat{f}(\cdot)$ to additionally minimize the discrepancy $\text{disc}_{\mathcal{H}}(\mathbb{P}_\mu(\phi(\mathbf{x})), \mathbb{P}_{\mu_{\text{OPT}}}(\phi(\mathbf{x})))$ between distributions of representations under the optimized distribution, $\mu_{\text{OPT}}$ and the training distribution $\mu$. This objective can be formalized as:

**Algorithm 1** IOM: Training and Optimization

1: **Input**: training data $\mathcal{D}$, number of gradient steps $T = 50$ to optimize $\mu_{\text{OPT}}$ starting from the training distribution $\mu$, training iteration $K$, batch size $n$,
2: **Initialize**: representation model $\phi_\eta(\cdot)$, learned model $\widehat{f}_\theta(\cdot)$, set of optimized design particles $\{\mathbf{x}_i^*\}_{i=1}^m$
3: **for** $k = 1 \cdots, K$ **do**,
4: Sample $n$ training points $(\mathbf{x}_i, y_i) \sim \mathcal{D}$
5: Run one gradient step (Equation 4) w.r.t $\theta_k$ and $\phi_k$ to obtain $\phi_{k+1}$ and $f_{\theta_{k+1}}(\phi_{k+1}(\cdot))$.
6: Run one gradient step to optimize design particles w.r.t. $f_\theta(\phi(\mathbf{x}))$: $\mathbf{x}_i^* \leftarrow \mathbf{x}_i^* + \eta \nabla_{\mathbf{x}} f_{\theta_k}(\phi_k(\mathbf{x}_i^*))$.
7: **end for**

$$\max_{\mu_{\text{OPT}}} \quad \mathbb{E}_{\widehat{\mathbf{x}}\sim\mu_{\text{OPT}}}\left[f_\theta^*(\phi^*(\widehat{\mathbf{x}}))\right]$$
$$\text{s.t.} \quad (\phi^*, f_\theta^*) = \arg\min_{\phi, \widehat{f}} \quad \frac{1}{n}\sum_{i=1}^{n}(\widehat{f}(\phi(\mathbf{x}_i)) - y_i)^2 + \lambda \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x})), \mathbb{P}_{\mu_{\text{OPT}}}(\phi(\mathbf{x}))), \quad (3)$$

where $\lambda$ denotes a weighting hyperparameter. In Section 4.2, we will describe how we can convert this bi-level optimization problem into a practical MBO algorithm that can be implemented with high capacity deep neural networks. This is followed by a theoretical analysis to formally justify

the objective in Section 4.3, and a discussion of how the hyperparameter $\lambda$ can be tuned against a validation set, in Section 5, without any online access to the ground truth function.

### 4.2 Optimizing the Bi-Level Problem and the Practical IOM Algorithm

In this section, we will describe how we can convert the abstract bi-level optimization problem above into an objective that can be trained practically and discuss some practical implementation details. IOM alternates between solving the bi-level optimization problem with respect to the distribution $\mu_{\text{OPT}}$ and the learned model $f_\theta$ and $\phi$ independently. For training the learned model and the representation $f_\theta$ and $\phi$, IOM solves the inner optimization by running one-step of gradient descent (GD), utilizing the current snapshot of $\mu_{\text{OPT}}$ (denoted $\mu^t_{\text{OPT}}$):

$$(\phi_{t+1}, \theta_{t+1}) = \mathsf{GD}_{\theta,\phi;t}\left(\frac{1}{n}\sum_{i=1}^{n}(f_\theta(\phi(\mathbf{x}_i)) - y_i)^2 + \lambda \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x})), \mathbb{P}_{\mu^t_{\text{OPT}}}(\phi(\mathbf{x})))\right). \quad (4)$$

Simultaneously, we update the design particles representing $\mu^t_{\text{OPT}} := \delta\{\mathbf{x}^1_t, \mathbf{x}^2_t, \cdots\}$ towards optimizing the learned function $f_{\theta_{t+1}}(\phi_{t+1}(\mathbf{x}))$: $\mathbf{x}^i_{t+1} \leftarrow \mathbf{x}^i_t + \eta \nabla_{\mathbf{x}} f_{\theta_{t+1}}(\phi_{t+1}(\mathbf{x}))|_{\mathbf{x}=\mathbf{x}^i_t}$, $\forall i$. Pseudocode for IOM is shown in Algorithm 1.

**Practical implementation details:** We model the representation $\phi(\mathbf{x})$ and the learned function $f_\theta(\cdot)$ each as two-hidden layer ReLU networks with sizes 2048 and 1024, respectively. We utilize the $\chi^2$-discrepancy, $\text{disc}_{\mathcal{H}}$ in our experiments: $\text{disc}_\phi(p, q) := \chi^2([\phi(\mathbf{x})]_{\mathbf{x}\sim p}, [\phi(\mathbf{x})]_{\mathbf{x}\sim q})$. This divergence can be instantiated in its variational form via a least-squares generative adversarial network (LS-GAN) [34] on the representations $\phi(\mathbf{x})$. Concretely, we train an LS-GAN discriminator to discriminate between the representations $\phi(\mathbf{x})$ under the training distribution $\mu(\mathbf{x})$ and the optimized distribution $\mu_{\text{OPT}}(\mathbf{x})$, whereas the feature representations under these distributions are optimized to be as invariant to each other as possible. We also evaluated a variety of discrepancy measures, including the maximum mean discrepancy (MMD) [16], but found the $\chi^2$-divergence to be the best performing version across the board. More implementation details can be found in Appendix A.2.

### 4.3 Theoretical Analysis of Invariant Representation Learning for MBO

In this section, we will formally show that under certain standard assumptions, solving the bi-level optimization problem (Equation 3) enables us to find a better design than the dataset distribution $\mu_{\text{data}}$, similar to analyses of offline RL algorithms. To show this, we will combine theoretical tools for analyzing offline RL algorithms [10, 28, 52] and domain adaptation [55]. We show that the expected value of the ground truth objective under any given distribution $\pi(\mathbf{x})$ of design inputs can be lower bounded in terms of the expected value under the learned function modulo some error terms. Then we will show that these error terms are controlled tightly when solving the bi-level problem, and finally appeal to uniform concentration to show that the performance of $\mu_{\text{OPT}}$ is better with high probability. Our goal isn't to derive the tightest possible bound for IOM, but show that IOM can attain reasonable performance guarantees, and this domain adaptation perspective can tackle distributional shift.

**Notation and assumptions.** Following standard assumptions [31], we will analyze the setting when the learned representation lies in some space of functions, $\phi \in \Phi$, and the learned function $f_\theta$ lies in $f_\theta \in \mathcal{F}$. We will consider that the optimized distribution $\mu_{\text{OPT}}$ belongs to a function class, $\mu_{\text{OPT}} \in \Pi$. Our analysis will require that the ground truth model $f$ is approximately realizable under some representation $\phi \in \Phi$ and for some $g \in \mathcal{F}$. That is, we assume:

**Assumption 4.1** ($\varepsilon-$realizability)**.** For any distribution $\pi$ over designs, there exists a representation $\phi \in \Phi$ and a $g \in \mathcal{F}$ such that $\min_g \max_{\pi\in\Pi} \mathbb{E}_{\mathbf{x}\sim\pi}\left[|f(\mathbf{x}) - g(\phi(\mathbf{x}))|\right] \leq \varepsilon_{\mathcal{F},\Phi}$.

In addition, we assume that $\forall f \in \mathcal{F}, ||f||_\infty < \infty$ and is Lipschitz continuous with constant $C$: $||f||_{\text{L}} \leq C$. This is true for most parameteric function classes, such as neural networks. Then, we can lower bound the average objective value under any distribution in terms of the invariance:

**Proposition 4.2** ((Informal) Lower-bounding the ground truth value under $\pi$)**.** *Under Assumption 4.1 and the Lipschitz continuity on the function $f_\theta$, the ground truth objective for any given $\pi \in \Pi$ can be lower bounded in terms of the learned objective model $f_\theta \in \mathcal{F}$ and representation $\phi \in \Phi$, with high probability $\geq 1 - \delta$ that:*

$$J(\pi) - J_\theta(\pi) \geq \underbrace{J(\mu_{data}) - J_\theta(\mu_{data})}_{(\blacksquare)} - \underbrace{C_{\mathcal{F}} \cdot disc_{\mathcal{H}}(\mathbb{P}_{\mu_{data}}(\phi(\mathbf{x})), \mathbb{P}_\pi(\phi(\mathbf{x})))}_{(*)} - \varepsilon_{\mathcal{F},\Phi} - \varepsilon_{stat},$$

*where $C_{\mathcal{F}}$ is a uniform constant depending on the function class $\mathcal{F}$ and $\varepsilon_{stat}$ refers to statistical error that decays inversely with the dataset size, $|\mathcal{D}|$.*

A proof for Proposition 4.2 is in Appendix A.1. The ■ term can further be lower bounded using a standard concentration error bound for ERM, and can be controlled if we train $f_\theta(\phi(\cdot))$ minimize prediction error against ground truth values of $f$. Now we utilize Proposition 4.2 and a uniform concentration argument to lower-bound the value under the optimized distribution $\mu_{\text{OPT}}$.

**Proposition 4.3** ((Informal) Performance guarantee for IOM)**.** *Under Assumption 4.1, the expected value of the ground truth objective under $\mu_{\text{OPT}}$, $J(\mu_{\text{OPT}})$ is lower bounded by:*

$$J(\mu_{\text{OPT}}) \gtrsim J(\mu_{data}) - \mathcal{O}\left(\sqrt{\frac{\log \frac{|\mathcal{F}||\Phi||\Pi|}{\delta}}{|\mathcal{D}|}} + \frac{\log \frac{|\mathcal{F}||\Phi||\Pi|}{\delta}}{|\mathcal{D}|}\right) + \underbrace{J_\theta(\mu_{\text{OPT}}) - J_\theta(\mu_{data})}_{(\circ)} - (*). \quad (5)$$

A proof of Proposition 4.3 can be found in Appendix A.1. This proposition implies that as long as the optimizer finds designs that improve the learned objective function $f_\theta \in \mathcal{F}$, such that $(\circ)$ is positive, and the discrepancy term $(\star)$ is minimized, then the distribution over designs found by IOM, $\mu_{\text{OPT}}$, will compete favorably against the distributions of designs in the dataset. Of course, these bounds present the worst-case guarantees for IOM, but as we will show in our experiments, IOM does clearly improve over the best designs in the offline dataset. In addition, this theoretical analysis also motivates the design of our scheme for performing hyperparameter tuning and model selection. This scheme will be based directly on the result from Proposition 4.2. We will discuss this connection when discussing tuning in Section 5.

Finally, we would like to highlight how invariance leads to "average" values if $\mu_{\text{OPT}}$ deviates too far away from the data distribution $\mu$. If $\mu_{\text{OPT}}$ deviates too far, such that $\text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{data}}}(\phi), \mathbb{P}_{\mu_{\text{OPT}}}(\phi)) \leq \varepsilon'$, the learned function is bounded too close to the average learned value on the data i.e., $|\widehat{J}(\mu_{\text{OPT}}) - \widehat{J}(\mu_{\text{data}})| \leq C' \cdot \varepsilon'$. Thus, the learned function simply reverts to "predicting the mean objective value" of the training distribution as $\varepsilon'$ decreases, thereby losing any information about the identity of the optimized distribution, when the optimizer goes too far out of $\mu$.

## 5 Offline Workflow and Tuning for IOM

A central challenge in offline model-based optimization is that of offline workflow and hyperparameter tuning: not only must an algorithm be able to produce designs using a static dataset, but it must also prescribe a *fully offline* workflow for tuning hyperparameters so that the produced designs are as good as possible. This includes both explicit hyperparameters (such as the coefficient of conservatism or invariance, $\lambda$ for IOM) and implicit hyperparameters (such as the early stopping point). The primary challenge stems from the fact that unlike supervised learning, the final performance in MBO is determined by the optimized distribution $\mu_{\text{OPT}}$, which is different from the training distribution, $\mu_{\text{data}}$, and we must reason about this performance fully offline.

In this section, we discuss how IOM admits a particularly convenient and *fully offline* hyperparameter tuning procedure that works well empirically. The goal of our tuning problem is to find a tuple consisting of the learned representation $\phi^\lambda$, the learned model, $f_\theta^\lambda$, and the optimized distribution $\mu_{\text{OPT}}^\lambda$, out of a given set of tuples obtained for different values of $\lambda$. In other words, we wish to find the $\lambda$ from the set of models $M_\lambda := (f_\theta^\lambda, \phi^\lambda, \mu_{\mu_{\text{OPT}}}^\lambda)$, such that the corresponding $\mu_{\text{OPT}}^\lambda$ attains the highest value under the ground truth objective. Crucially note that this must be done fully offline.

**Our key idea:** The primary question we wish to answer is: how can we identify if a given tuple $(f_\theta^\lambda, \phi^\lambda, \mu_{\text{OPT}}^\lambda)$ leads to good performance? In order for a tuple trained via IOM to perform well, we need two primary conditions: **(1)** it should be nearly invariant, and as a result, robust to distributional shift, and **(2)** the learned $f_\theta^\lambda$ must attain good generalization performance within the training distribution. Intuitively **(1)** guards us against distributional shift, and **(2)** enables us to find as good of a learned model on the dataset, that does not overfit or underfit as a result of the invariance regularizer. This forms the basis of our tuning method.

**Offline tuning of IOM:** To instantiate the idea practically, we first run IOM with a set of hyperparameters $\Lambda = \{\lambda_1, \lambda_2, \cdots\}$. Then for each run, we record the validation in-distribution error and the value of the invariance regularizer on a validation set. We then perform tuning in two steps: first, we filter all $\lambda$ values that attain near-perfect invariance under a user-specified margin $\varepsilon$, i.e., $\text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{data}}}(\phi^\lambda(\mathbf{x})), \mathbb{P}_{\mu_{\text{OPT}}^\omega}(\phi^\lambda(\mathbf{x}))) \leq \epsilon$, and are hence guaranteed to be robust to distributional shift. Second, we now pick models that attain good performance within the training distribution by selecting $\lambda$ values that attain the smallest validation prediction error: $(f_\theta(\phi(\mathbf{x})) - y)^2$, in addition to picking

the early stopping point based on the smallest validation error. This enables us to pick $\lambda$ that gives rise to a generalizing model $f_\theta^\lambda, \phi^\lambda$, while being most robust to distributional shift. This still leaves a user-defined margin $\varepsilon$ as a free parameter, but we find that it is comparatively insensitive and can be selected heuristically, as we show in our experiments in Section 6.2. Formally, this procedure corresponds to solving the following optimization:

$$\lambda^* := \arg\min_{\lambda \in \Lambda} \; \frac{1}{n} \sum_{i=1}^{n_{\text{val}}} (f_\theta^\lambda(\phi^\lambda(\mathbf{x}_i)) - y_i)^2 \;\; \text{s.t.} \;\; \text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{val}}}(\phi^\lambda(\mathbf{x})), \mathbb{P}_{\mu^\lambda_{\text{OPT}}}(\phi^\lambda(\mathbf{x}))) \leq \epsilon \tag{6}$$

Finally, for a more formal explanation of the soundness of this strategy, we note that applying Equation 6 provides a tight lower bound on the ground truth objective value in Proposition 4.2. Up to statistical error (which decays as $|\mathcal{D}|$ increases), the prediction error on a held-out validation set estimates ($\blacksquare$) in Proposition 4.2, and a smaller discrepancy (i.e., $\text{disc}_{\mathcal{H}} \leq \varepsilon$) controls the ($\star$) term in Prop. 4.2. $\varepsilon_{\text{stat}}$ is statistical error, and $\varepsilon_{\mathcal{F},\Phi}$ only depends on the function classes involved ($\mathcal{F}, \Phi$). This tuning $\lambda$ per Equation 6 can be justified as aiming to maximize a lower bound on $J(\mu_{\text{OPT}})$.

**Practical implementation.** When implementing this procedure in practice, we must first select the value of $\varepsilon$. For obtaining perfect invariance, the value of $\varepsilon$ must be equal to 0.25 (when the discriminator is trained using a $\chi^2$-divergence least-squares GAN objective), and therefore, $\varepsilon$ must be set close to 0.25. However, in practice, instead of selecting $\varepsilon$, we can select a fraction of all possible $\lambda$ values that pass the filter. In our experiments, we sort the runs based on the absolute difference between the value of invariance regularizer (i.e., the discriminator training objective) and 0.5, and let the top 15% (top 3 out of the 7 $\lambda$ values we tune on) of the runs be selected for the second step.

In Section 6, we validate this offline tuning strategy, both against other alternative offline strategies for tuning IOM as well as offline strategies for tuning prior methods (COMs [46] and gradient descent [45]). We find that our offline tuning strategy performs comparably to full online tuning without needing any online queries, whereas other strategies for IOM and other methods, lead to significantly poor performance compared to online tuning.

# 6 Experimental Evaluation

The goal of our empirical evaluation is to answer the following questions: **(1)** Does IOM outperform prior state-of-the-art methods for data-driven model-based optimization? **(2)** Is our offline tuning strategy for IOM better than other offline tuning strategies for IOM or other prior methods? **(3)** How sensitive is IOM to the choice of the hyperparameter $\lambda$? In this section, we will answer these questions via a comparative evaluation of IOM on several continuous data-driven design tasks from the standard tasks from the Design-Bench suite [45].

## 6.1 Empirical Evaluation on Benchmark Tasks

We compare IOM to a variety of prior methods for model-based optimization: CbAS [8], Auto-focused CbAS [11] and MINs [26] that utilize generative models for constraining distributional shift; COMs [46] and RoMA [51] train robust models of the objective function by via conservatism and smoothness, respectively; gradient-free optimization methods such as REINFORCE [48], CMA-ES [18] and conventional Bayesian optimization BO-qEI [49]. We also compare the standard approach of learning a naïve model of the objective function, which is then optimized via gradient ascent. To better understand the benefits of enforcing invariance, we also compare IOM to IOM-C, which additionally applies a conservatism term on top of invariance that pushes up the learned values of designs in the dataset. Additionally, we add a prior method that optimizes designs against a lower-confidence bound estimate computed using a Gaussian process posterior in Appendix A.3.

**Tasks.** We evaluate on four tasks with continuous-valued input space from the Design-Bench [45] benchmark for offline model-based optimization (MBO). Here we briefly summarize these tasks: **(1)** Superconductor: this task requires optimizing the properties of superconducting materials to maximize the operating temperature; the dataset consists of 17010 training points, and each input $\mathbf{x} \in \mathbb{R}^{86}$ represents a serialization of the chemical formula. **(2)** D'Kitty and **(3)** Ant robot morphology design: these tasks require selecting the optimal morphology of a quadrupedal robot, in order to maximize how quickly it can walk or crawl in a particular direction; the morphologies are represented by $\mathbf{x} \in \mathbb{R}^{56}$ and $\mathbf{x} \in \mathbb{R}^{60}$, respectively. **(4)** Hopper controller: this task reframes the classic hopper task [7] as a bandit problem, where instead of sequentially controlling the hopper, the goal is to directly output the parameters of a neural network controller that maximizes hopping speed; note that this task, as set up in Design-Bench, is *not* a RL problem, but rather a bandit problem with a 5126-dimensional continuous action space corresponding to neural network weights.

**Evaluation protocol.** Following prior work [45, 46, 51, 12], for each method, we query the learned model to obtain the top performing $N$ points, i.e., $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$. We report the maximum objective value among these $N$ samples. This protocol reasonably reflects a real-world design process, where a number of computationally produced designs are tested and the best one is used for deployment.

**Hyperparameter tuning.** For fair comparison against prior methods studied in Trabucco et al. [45], we follow the uniform tuning strategy for reporting results in this section: we run IOM on each task for a given set of $\lambda$ values, and pick a single value of $\lambda$ across every task. Note that this does require access to online evaluation, but prior works do use the same protocol. We will evaluate the efficacy of our offline tuning approach in Section 6.2 extensively.

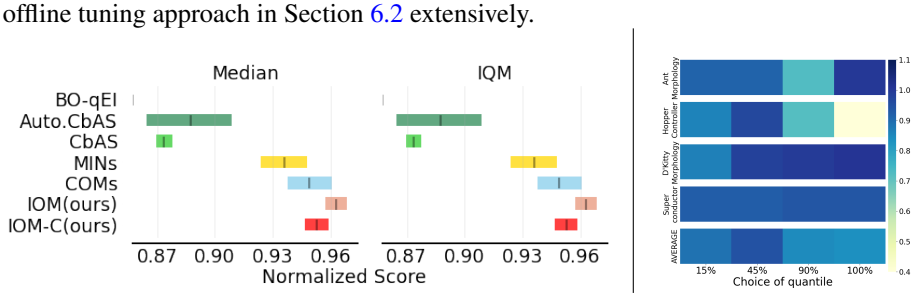


Figure 2: **Left**: Median and IQM [1] (with 95% Stratified Bootstrap CIs) for the aggregated normalized score on Design-bench. IOM improves over the prior methods, including those which use explicit conservatism although IOM does not. As we could not find the individual runs for RoMA [51], we report the mean and standard deviation for RoMA (copied from the results in Yu et al. [51]) in Appendix A.3. Other baselines including comparisons to best in the dataset are in Appendix A.3. **Right**: The offline tuning performance matrix under different quantiles to filter the models based on discriminator loss, compared with the oracle uniform tuning, which shows our tuning strategy is robust to this hyper-parameter.

**Results.** Following the convention proposed by Trabucco et al. [45], we evaluate all methods in terms of normalized score (higher is better). In Figure 2 (left), we show the median and interquantile mean (IQM) [1] for the aggregated scores across all tasks. Due to space limits, we only report the scores for the top-performing baselines (See Appendix A.2 for the results for all baselines). IOM significantly outperforms all of the prior methods, including state-of-the-art methods that employ conservatism, such as COMs [46]. This indicates that invariance alone (IOM) can effectively handle distribution shifts and avoid over-estimation for out-of-distribution actions. Additionally, a comparison between the performance of IOM and IOM with conservatism (IOM-C) suggests that adding conservatism does not improve performance further. This further corroborates the fact that invariance is effective in preventing the optimizer from going out of distribution and conservatism is not needed. That said, we would like to clarify that this result does not mean that invariance will always be better than conservatism on any given offline MBO problem. In order to understand why IOM outperforms COMs on these tasks, we examine the in-distribution validation error of IOM and COMs in Appendix A.6 and find that IOM learns less distorted functions that generalize better.

### 6.2 Analysis of the Offline Workflow for Tuning IOM

We will now present experiments to understand the effectiveness of our offline tuning scheme (Section 5) for tuning IOM. It is important to highlight that, in general, offline tuning presents significant challenges for MBO. Prior works in bandits and reinforcement learning [54] have suggested that offline tuning should not work at all and, to the best of our knowledge, no prior work in offline MBO has proposed a principled and effective offline tuning method. In this section, we empirically validate our tuning strategy in comparison to other potential strategies, and compare results from our approach to an oracle strategy that uses online evaluation of the true function.

**Comparisons.** We compare our offline tuning strategy to various offline strategies: (1). only tunes IOM based on validation-set prediction error, analogous to supervised learning (called "prediction error only"); (2). a strategy that tunes IOM based on only the value of the invariance regularizer (called "invariance only"); (3). a method that utilizes our tuning strategy for picking $\lambda$, but not for picking the checkpoint ("uniform checkpoint selection") and two offline tuning strategies for tuning other prior methods: (a) tuning COMs [46] using validation-set prediction error (called "prediction error only"), conservatism loss (called "conservatism loss only") and both prediction error and conservatism loss, one by one (called "two-step tuning"); and (b) tuning an objective model trained via gradient ascent [45] using validation MSE error ("gradient ascent"). We report the drop in performance relative to the corresponding oracle online strategy that actually computes the "groundtruth" values of optimized designs in the simulator to choose hyperparameters, and is therefore not fully offline,

i.e., the relative drop in performance of the method (IOM, COM, grad. ascent) when tuned offline using the particular strategy compared to oracle tuning for the same method.

**Results.** We present our results in Figure 3 (right) comparing the efficacy of various tuning schemes across various methods averaged over all tasks. While offline tuning does lead to a reduction in performance compared to oracle online tuning, we find that the drop is smallest for our offline tuning strategy applied to IOM. All other tuning strategies based on either only prediction error or the invariance regularizer lead to worse performance. This indicates that our tuning strategy is effective in tuning IOM. Furthermore, offline tuning strategies that utilize a validation set with prior methods, such as COMs and gradient ascent, also lead to a significant decrease in performance relative to the oracle. This is quite appealing as it suggests that IOM is more amenable to offline tuning strategies relative to prior offline MBO algorithms. Even if these prior methods can perform well when provided access to online evaluations, they might start to fail in real-world MBO problems when one needs to tune them fully offline. We discuss computational complexity of tuning the methods in Appendix A.6.

**Ablation analyses.** Next we aim to study the sensitivity of our offline tuning strategy with respect to the non-rigorous, user-specified parameter $\varepsilon$ in Equation 6 and checkpoint selection based on the validation prediction error. Figure 2 (right) shows the offline tuning performance heatmap for all tasks (along their aggregated performance in the bottom row) under different choice of quantiles (where quantile $x\%$ means the top $x\%$ of the models according to the value of the invariance regularizer, and the given value of $\epsilon$ were chosen). The color in each position of the matrix corresponds to the ratio of performance of our offline tuning strategy for a given quantile value, compared to that of uniform oracle tuning. As shown in the bottom row, aggregated over all tasks, we can see our offline selection strategy is quite robust to the hyperparameter $\epsilon$. Figure 3 (left) evaluates the efficacy of our checkpoint selection strategy, we plot the objective value $J(\mu_{\text{OPT}})$ attained as a function of the number of training epochs (i.e., checkpoints), alongside the values for the prediction MSE in the validation set for the corresponding epochs. The checkpoints selected by our early stopping scheme are shown via the vertical line. Observe that indeed the checkpoint for the lowest validation MSE corresponds to a rather good optimized distribution $\mu_{\text{OPT}}$ for different tasks, no matter the optimal training epochs happen at the beginning or near the end, which validates our early stopping strategy. More detailed results showing the efficacy of early stopping for both IOM can be found in Appendix A.3.

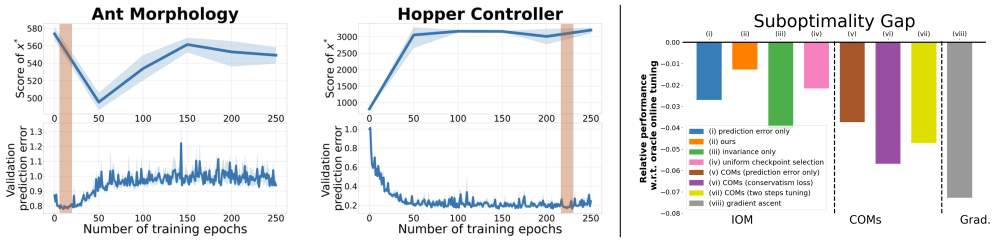


Figure 3: **Left**: Visualizing our checkpoint selection scheme on two tasks, and no matter the best checkpoint happens at the beginning or end of the training, our strategy based on validation in-distribution MSE could capture the best performance. **Right**: Offline tuning vs oracle uniform online tuning for IOM, COMs and Gradient-ascent. Among all the methods, IOM achieves the smallest regret for offline tuning, and our specific tuning strategy works best compared with other tuning strategies for IOM.

## 7 Discussion

In this work we study offline data-driven model based optimization (MBO), where the goal is to produce optimized designs, purely using a static dataset. We proposed to view offline MBO through the lens of domain adaptation, and proposed a method that trains a model of the objective with an additional regularizer to promote invariance between the representations learned by the model in expectation under the training data distribution and the distribution of optimized designs. We show that this leads to a practically effective data-driven optimization method, and an appealing workflow for selecting the hyperparameters with offline data. Our method outperforms prior approaches, which illustrates the practical benefits of invariance. We also validate the effectiveness of the accompanying tuning strategy. While we evaluate our method on MBO problems, our approach can in principle also be extended to contextual bandits and even to, sequential reinforcement learning. **Limitations**: **(1)** fully data-driven offline optimization may on its own be insufficient (due to limitations of staying close to the dataset, hardness of tuning) and inevitably will require access to online evaluation. **(2)** a full understanding of theoretical conditions when invariance can outperform conservatism is important, but out of the scope of this work.

# References


[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[2] Christof Angermueller, David Belanger, Andreea Gane, Zelda Mariet, David Dohan, Kevin Murphy, Lucy Colwell, and D Sculley. Population-based black-box optimization for biological sequence design. *arXiv preprint arXiv:2006.03227*, 2020.

[3] Christof Angermueller, David Dohan, David Belanger, Ramya Deshpande, Kevin Murphy, and Lucy Colwell. Model-based reinforcement learning for biological sequence design. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HklxbgBKvr.

[4] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1): 151–175, 2010.

[5] Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. Safe controller optimization for quadrotors with gaussian processes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 491–496. IEEE, 2016.

[6] Surojit Biswas, Grigory Khimulya, Ethan C Alley, Kevin M Esvelt, and George M Church. Low-n protein engineering with data-efficient deep learning. *Nature methods*, 18(4):389–396, 2021.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[8] David Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019. URL http://proceedings.mlr.press/v97/brookes19a.html.

[9] David H Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. *arXiv preprint arXiv:1901.10060*, 2019.

[10] Ching-An Cheng, Tengyang Xie, Nan Jiang, and Alekh Agarwal. Adversarially trained actor critic for offline reinforcement learning. *arXiv preprint arXiv:2202.02446*, 2022.

[11] Clara Fannjiang and Jennifer Listgarten. Autofocused oracles for model-based design. *arXiv preprint arXiv:2006.08052*, 2020.

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

[13] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[14] Anna Gaulton, Louisa J Bellis, A Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, et al. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1):D1100–D1107, 2012.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. NIPS'14, 2014.

[16] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[17] Kam Hamidieh. 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.

[18] Nikolaus Hansen. The CMA evolution strategy: A comparing review. In José Antonio Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea, editors, *Towards a New Evolutionary Computation - Advances in the Estimation of Distribution Algorithms*, volume 192 of *Studies in*

*Fuzziness and Soft Computing*, pages 75–102. Springer, 2006. doi: 10.1007/3-540-32494-1_4. URL https://doi.org/10.1007/3-540-32494-1_4.

[19] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[20] Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is pessimism provably efficient for offline rl? *arXiv preprint arXiv:2012.15085*, 2020.

[21] Fredrik Johansson, Uri Shalit, and David Sontag. Learning representations for counterfactual inference. In *International conference on machine learning*, pages 3020–3029. PMLR, 2016.

[22] Fredrik D Johansson, Nathan Kallus, Uri Shalit, and David Sontag. Learning weighted representations for generalization across designs. *arXiv preprint arXiv:1802.08598*, 2018.

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL http://arxiv.org/abs/1312.6114. cite arxiv:1312.6114.

[24] Johannes Kirschner and Andreas Krause. Bias-robust bayesian optimization via dueling bandits. In *International Conference on Machine Learning*, pages 5595–5605. PMLR, 2021.

[25] Johannes Kirschner, Ilija Bogunovic, Stefanie Jegelka, and Andreas Krause. Distributionally robust bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 2174–2184. PMLR, 2020.

[26] Aviral Kumar and Sergey Levine. Model inversion networks for model-based optimization. *NeurIPS*, 2019.

[27] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.

[28] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[29] Aviral Kumar, Amir Yazdanbakhsh, Milad Hashemi, Kevin Swersky, and Sergey Levine. Data-driven offline optimization for architecting hardware accelerators. *arXiv preprint arXiv:2110.11346*, 2021.

[30] Romain Laroche, Paul Trichelair, and Rémi Tachet des Combes. Safe policy improvement with baseline bootstrapping. *arXiv preprint arXiv:1712.06924*, 2017.

[31] Byung-Jun Lee, Jongmin Lee, and Kee-Eung Kim. Representation balancing offline model-based reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=QpNz8r_Ri2Y.

[32] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[33] Yao Liu, Omer Gottesman, Aniruddh Raghu, Matthieu Komorowski, Aldo Faisal, Finale Doshi-Velez, and Emma Brunskill. Representation balancing mdps for off-policy policy evaluation. *arXiv preprint arXiv:1805.09044*, 2018.

[34] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[35] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

[36] Willie Neiswanger and Aaditya Ramdas. Uncertainty quantification using martingales for misspecified gaussian processes. In *Algorithmic Learning Theory*, pages 963–982. PMLR, 2021.

[37] Quoc Phong Nguyen, Zhongxiang Dai, Bryan Kian Hsiang Low, and Patrick Jaillet. Value-at-risk optimization with gaussian processes. In *International Conference on Machine Learning*, pages 8063–8072. PMLR, 2021.

[38] Thanh Nguyen-Tang, Sunil Gupta, A Tuan Nguyen, and Svetha Venkatesh. Offline neural contextual bandits: Pessimism, optimization and generalization. *arXiv preprint arXiv:2111.13807*, 2021.

[39] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[41] Thiago D Simão, Romain Laroche, and Rémi Tachet des Combes. Safe policy improvement with an estimated baseline policy. *arXiv preprint arXiv:1909.05236*, 2019.

[42] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, 2012. URL http://dl.acm.org/citation.cfm?id=2999325.2999464.

[43] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, 2015.

[44] Bharath K Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernhard Schölkopf, and Gert RG Lanckriet. On integral probability metrics,\phi-divergences and binary classification. *arXiv preprint arXiv:0901.2698*, 2009.

[45] Brandon Trabucco, Xinyang Geng, Aviral Kumar, and Sergey Levine. Design-bench: Benchmarks for data-driven offline model-based optimization, 2021. URL https://github.com/brandontrabucco/design-bench.

[46] Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Conservative objective models for effective offline model-based optimization. In *International Conference on Machine Learning*, pages 10358–10368. PMLR, 2021.

[47] Martin J Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge University Press, 2019.

[48] Ronald J. Williams. 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.

[49] James T. Wilson, Riccardo Moriconi, Frank Hutter, and Marc Peter Deisenroth. The reparameterization trick for acquisition functions. *CoRR*, abs/1712.00424, 2017. URL http://arxiv.org/abs/1712.00424.

[50] Liuyi Yao, Sheng Li, Yaliang Li, Mengdi Huai, Jing Gao, and Aidong Zhang. Representation learning for treatment effect estimation from observational data. *Advances in Neural Information Processing Systems*, 31, 2018.

[51] Sihyun Yu, Sungsoo Ahn, Le Song, and Jinwoo Shin. Roma: Robust model adaptation for offline model-based optimization. *Advances in Neural Information Processing Systems*, 34, 2021.

[52] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[53] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *arXiv preprint arXiv:2102.08363*, 2021.

[54] Andrea Zanette. Exponential lower bounds for batch reinforcement learning: Batch rl can be exponentially harder than online rl. In *International Conference on Machine Learning*, pages 12287–12297. PMLR, 2021.

[55] Han Zhao, Remi Tachet Des Combes, Kun Zhang, and Geoffrey Gordon. On learning invariant representations for domain adaptation. In *International Conference on Machine Learning*, pages 7523–7532. PMLR, 2019.

# Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We discuss limitations of our work in Section 7.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our work pertains to fundamental algorithm design for model-based optimization, and it does not have any particular negative societal impacts beyond those that apply more generally to model-based optimization methods.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] We discuss the full set of assumptions in Section 4.3.
   (b) Did you include complete proofs of all theoretical results? [Yes] Our proofs are included in Appendix A.1.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include it with the supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We include these details in Appendix A.2.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We reported error bars in our results, and also followed guidelines for rigorous statistical evaluation using the protocol from Agarwal et al. [1].
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] These details can be found in Appendix A.2.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited Design Bench.
   (b) Did you mention the license of the assets? [Yes] Details are found in Appendix A.2.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

# A Appendices

## A.1 Proofs of Theoretical Results

In this section, we will provide proofs of the various theoretical results from Section 4.3. We will first discuss the assumptions and technical conditions we use, then prove the lower bound result from Proposition 4.2 and finally utilize it to prove the performance guarantee for invariant representation learning (Proposition 4.3). Before diving into the proofs, we first recall the set of assumptions and technical conditions below:

### A.1.1 Assumptions, Technical Conditions and Lemmas

Recall that our proofs operate under the following assumptions:

- **$\varepsilon$-realizability:** For any distribution $\pi$ over designs, there exists a representation $\phi \in \Phi$ and a $g \in \mathcal{F}$ such that $\min_{\phi,g} \max_{\pi \in \Pi} \mathbb{E}_{\mathbf{x}\sim\pi} \left[|f(\mathbf{x}) - g(\phi(\mathbf{x}))|\right] \leq \varepsilon_{\mathcal{F},\Phi}$.
- $\forall f \in \mathcal{F}, ||f||_\infty < \infty$.
- $\forall f \in \mathcal{F}, ||f||_{\mathrm{L}} \leq C$.

Additionally, we will also define statistical error that will appear in the bounds:

$$\varepsilon_{\text{stat}} \approx \sqrt{\frac{\log\left(\frac{|\mathcal{F}||\Phi|c_0}{\delta}\right)}{|\mathcal{D}|}} \tag{7}$$

We will use some helpful Lemmas that we list below, which can be proven using standard results from empirical risk minimization and concentration inequalities [47].

**Lemma A.1** (Concentration of expected function values)**.** *For any given $f \in \mathcal{F}$ satisfying the assumptions above, and any given distribution $\pi$ over designs $\mathcal{X}$, let $J_f(\pi) := \mathbb{E}_{\mathbf{x}\sim\pi}[f(\mathbf{x})]$. Then, given a set of $N$ samples, $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N \sim \pi$, the empirical mean $\widehat{J}_f(\pi) = \frac{1}{N}\sum_{i=1}^N f(\mathbf{x}_i)$ converges to $J_f(\pi)$ such that, with high probability $\geq 1-\delta$ ($c_0$ is a universal constant),*

$$\left|J_f(\pi) - \widehat{J}_f(\pi)\right| \lesssim \sqrt{\frac{\log\left(\frac{c_0}{\delta}\right)}{N}}. \tag{8}$$

Building on Lemma A.1, we now bound the empirical risk and the population risk of a given loss function $\mathcal{L}(f)$ at the empirical risk minimizer using a standard uniform concentration argument:

**Lemma A.2** (Concentration of empirical risk minimizer)**.** *Given any bounded, Lipschitz loss function, $\mathcal{L}$ and for a given $f \in \mathcal{F}$, let, $\mathcal{L}(f) := \mathbb{E}_{\mathbf{x}\sim\mu_{data}}[L(f,\mathbf{x})]$, and let the corresponding empirical loss function be denoted by: $\widehat{\mathcal{L}}(f) := \frac{1}{|\mathcal{D}|}\sum_i L(f,\mathbf{x}_i)$. Then, with high probability $\geq 1-\delta$,*

$$\left|\mathcal{L}\left(\arg\inf_{f\in\mathcal{F}} \widehat{\mathcal{L}}(f)\right) - \mathcal{L}\left(\arg\inf_{f\in\mathcal{F}} \mathcal{L}(f)\right)\right| \lesssim \mathcal{O}\left(||\mathcal{L}||_\infty \sqrt{\frac{1}{|\mathcal{D}|}\log\left(\frac{|\mathcal{F}|c_0}{\delta}\right)}\right). \tag{9}$$

Finally, we discuss a technical condition that will be required to prove Proposition 4.2. We consider the following "continuity" property of the function class $\mathcal{F}$ with respect to a loss function that is a mixture of two loss functions:

**Definition A.3** (Continuity of $\mathcal{F}$ with respect to $\mathcal{L}_\lambda$.)**.** We say that a function class $\mathcal{F}$ is continuous with respect to a given loss function $\mathcal{L}_\lambda(f) := \mathcal{L}_1(f) + \lambda\mathcal{L}_2(f)$, where $\lambda \in \mathbb{R}^+$ is a hyperparameter, when the following holds: denoting $f_\theta^\lambda := \inf_{f\in\mathcal{F}} \mathcal{L}_\lambda(f)$, the $\lim_{\lambda\to 0} f_\theta^\lambda$ exists, and corresponds to a minimizer of $\mathcal{L}_1(f)$:

$$\lim_{\lambda\to 0} \; f_\theta^\lambda \in \arg\inf_{f\in\mathcal{F}} \mathcal{L}_1(f). \tag{10}$$

Definition A.3 formalizes the intuition that an addition of the loss $\mathcal{L}_2$ affects the optimal solution in a continuous manner, and does not abruptly change the solution. We would expect such a criterion to

be satisfied when optimizing smooth loss functions (e.g., squared error) over parameters dictated by a neural network, with appropriate explicit regularization (e.g., $\ell_2$ regularization on parameters) or when the training procedure smoothens the solution using some form of implicit regularization.

We will assume that the function class obtained by composing $\mathcal{F}$ and $\Phi$, $\mathcal{F} \circ \Phi$, obeys the continuity assumption with respect to the training objective in Equation 3. Concretely, in this case $\mathcal{L}_1$ and $\mathcal{L}_2$ are given by: $\mathcal{L}_1(f, \phi) = \mathbb{E}_{\mathbf{x},y}[(f(\phi(\mathbf{x})) - y)^2]$ and $\mathcal{L}_2(f, \phi) = \text{disc}_{\mathcal{H}}(\mathbb{P}_\pi(\phi(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x})))$, and both of the loss functions are smooth with respect to the functions $\phi$ and $f$. In our experiments, we utilized a square loss for the discrepancy $\text{disc}_{\mathcal{H}}$, and therefore, it is smooth. Definition A.3 in this case would reduce to the following condition:

**Assumption A.4** (Definition A.3 applied to $\mathcal{F} \circ \Phi$.)**.** For any given threshold $\delta \in \mathbb{R}$ such that $|\lambda| \leq \delta$, there exists a non-decreasing function $\zeta : \mathbb{R}^+ \to \mathbb{R}^+$ of $\lambda$ such that, $\zeta(0) = 0$ and $||f_\theta^\lambda \circ \phi_\theta^\lambda - f^* \circ \phi^*||_\infty \leq \zeta(\lambda)$ for some $(f^*, \phi^*) \in \arg\inf_{f \in \mathcal{F}, \phi \in \Phi} \mathcal{L}_1(f, \phi)$.

Essentially, Assumption A.4 suggests that if we can utilize an appropriately chosen value of $\lambda$, then we can roughly match the solution obtained by just minimizing $\mathcal{L}_1$, which is the prediction error in our case. This would allow us to control the discrepancy of a solution obtained by only optimizing the prediction error.

### A.1.2 Proof of Proposition 4.2

**Proposition A.5** (Lower bounding the value under $\pi$.)**.** *Under the assumptions and criteria listed above, the ground truth objective for any given $\pi \in \Pi$ can be lower bounded in terms of the learned objective model $f_\theta \in \mathcal{F}$ and representation $\phi \in \Phi$, with high probability $\geq 1 - \delta$ as:*

$$J(\pi) - J_\theta(\pi) \geq \underbrace{J(\mu_{data}) - J_\theta(\mu_{data})}_{(\blacksquare)} - \underbrace{C_{\mathcal{F}} \cdot disc_{\mathcal{H}}(\mathbb{P}_{\mu_{data}}(\phi(\mathbf{x})), \mathbb{P}_\pi(\phi(\mathbf{x})))}_{(*)} - 2\varepsilon'_{\mathcal{F},\Phi} - \varepsilon_{stat},$$

*where $C_{\mathcal{F}}$ and $\varepsilon'_{\mathcal{F},\phi}$ are universal constants that only depend on the function class $\mathcal{F}, \Phi$ and other universal constants associated with the loss functions, and $\varepsilon_{stat}$ is a statistical error term that decreases as the dataset size $|\mathcal{D}|$ increases.*

*Proof.* Our high-level strategy would be to first decompose the terms into different components, and bound these components separately. We begin with the following decomposition:

$$\begin{aligned}
&J(\pi) - J_\theta(\pi) && (11)\\
&= \mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f(\mathbf{x}))\right] - \mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f_\theta(\phi(\mathbf{x})))\right] && (12)\\
&= \underbrace{\mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}(\mathbf{x})}\left[f(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}(\mathbf{x})}\left[f_\theta(\phi(\mathbf{x}))\right]}_{(a)} && (13)\\
&\quad + \underbrace{\mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f(\mathbf{x}))\right] - \mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f_\theta(\phi(\mathbf{x})))\right] - \left(\mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}(\mathbf{x})}\left[f(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}(\mathbf{x})}\left[f_\theta(\phi(\mathbf{x}))\right]\right)}_{:=\Delta(\pi,\mu_{\text{data}})}.
\end{aligned}$$

Note that term (a) exactly corresponds to the first term $\blacksquare = J(\mu_{\text{data}}) - J_\theta(\mu_{\text{data}})$ in the expression in the proposition. To obtain the rest of the terms, we will now lower bound the remainder $\Delta(\pi, \mu_{\text{data}})$. First note that we can rearrange this term into a more convenient form:

$$\Delta(\pi, \mu_{\text{data}}) := -\underbrace{\left(\mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f_\theta(\phi(\mathbf{x}))\right] - \mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}(\mathbf{x})}\left[[f_\theta(\phi(\mathbf{x}))]\right]\right)}_{(i)} + \underbrace{\left(\mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}(\mathbf{x})}\left[f(\mathbf{x})\right]\right)}_{(ii)}. \tag{14}$$

Term (i) in the above equation can directly be bounded to give rise to one of the terms that contributes to the discrepancy term the bound as shown below:

$$\begin{aligned}
(i) &:= -\int_{\mathbf{x}} (\pi(\mathbf{x}) - \mu_{\text{data}}(\mathbf{x})) \cdot f_\theta(\phi(\mathbf{x}))\, d\mathbf{x} && (15)\\
&= -\int_{\mathbf{z}} (\pi(\mathbf{z}) - \mu_{\text{data}}(\mathbf{z})) \cdot f_\theta(\mathbf{z})\, d\mathbf{z} && (16)\\
\implies (i) &\leq ||\widehat{f}||_{\text{L}} \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_\pi(\phi(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x}))), && (17)
\end{aligned}$$

where the second step (Equation 16) follows by reparameterizing the first step (Equation 15) in terms of the representation $\mathbf{z} := \phi(\mathbf{x})$ and applying probability density change of variables simultaneously with the change of variables for integration, which leads to a cancellation of the Jacobian terms. $\mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x}))$ denotes the probability distribution of the representations $\phi(\mathbf{x})$ when $\mathbf{x} \sim \mu_{\text{data}}(\mathbf{x})$, and $\mathbb{P}_{\pi}(\phi(\mathbf{x}))$ denotes the probability distribution of the representation $\phi(\mathbf{x})$ when $\mathbf{x} \sim \pi(\mathbf{x})$. The last step follows from the fact that the discrepancy measure, $\text{disc}_{\mathcal{H}}(p, q)$, which is an integral probability metric (IPM) [44], upper bounds the total-variation divergence between $p$ and $q$.

Now we tackle term (ii). First note that under the $\varepsilon$-realizability assumption, we note that there exists a $g \in \mathcal{F}, \phi^* \in \Phi$ such that $f(\mathbf{x}) \approx g(\phi^*(\mathbf{x}))$ in expectation under $\mu_{\text{data}}$ and $\pi$ (Assumption 4.1 holds for any distribution $\Pi$). That is,

$$
\begin{aligned}
(ii) &\leq \left|\mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[g(\phi^*(\mathbf{x}))\right] - \mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}(\mathbf{x})}\left[g(\phi^*(\mathbf{x}))\right]\right| + 2\varepsilon_{\mathcal{F},\Phi} && (18)\\
&\leq ||g||_{\text{L}} \left|\left|\mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[\phi^*(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim\mu_{\text{data}}(\mathbf{x})}\left[\phi^*(\mathbf{x})\right]\right|\right| + 2\varepsilon_{\mathcal{F},\Phi} && (19)\\
&\leq ||g||_{\text{L}} \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi^*(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi^*(\mathbf{x}))) + 2\varepsilon_{\mathcal{F},\Phi}, && (20)
\end{aligned}
$$

where the second step follows from the Lipschitzness of $\mathcal{F}$ and the third step follows similar to Equation 17.

Finally, we will show that for an appropriately chosen $\lambda > 0$, $\text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi^*(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi^*(\mathbf{x})))$ is close to $\text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x})))$. Applying Assumption A.4, we get that, for any given $\lambda$, there exists a $\zeta$ such that:

$$
\begin{aligned}
&||f_{\theta}^{\lambda} \cdot \phi^{\lambda} - f^* \cdot \phi^*||_{\infty} \leq \zeta(\lambda) && (21)\\
\implies &||\mathcal{L}_2(f_{\theta}^{\lambda}, \phi^{\lambda}) - \mathcal{L}_2(f^*, \phi^*)||_{\infty} \leq C_{\mathcal{L}_2} \cdot \zeta(\lambda), && (22)
\end{aligned}
$$

where the second step follows from the fact that the loss functions, $\mathcal{L}_1$ and $\mathcal{L}_2$ (in this case, $\text{disc}_{\mathcal{H}}$ and $(\cdot)^2$) are smooth functions of their arguments, and $C_{\mathcal{L}_2} := C_{\mathcal{H}}$ is the coefficient of smoothness when $\mathcal{L}_2 = \text{disc}_{\mathcal{H}}$. Thus, we get that:

$$
\left|\text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi^{\lambda}(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi^{\lambda}(\mathbf{x}))) - \text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi^*(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi^*(\mathbf{x})))\right| \leq C_{\mathcal{H}} \cdot \zeta(\lambda). \quad (23)
$$

Finally, we need to bound $\text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi^{\lambda}(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi^{\lambda}(\mathbf{x})))$ in terms of the discrepancy value for the $\phi$ learned during training. Note that $\phi$ is obtained by minimizing the objective in Equation 3, which is the sample-based version of the objective used for obtaining $f^{\lambda}$. Using the uniform concentration argument from Lemma A.2, we can upper bound $\mathcal{L}(f^{\lambda}, \phi^{\lambda})$ in terms of $\mathcal{L}(f_{\theta}, \phi)$:

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x},y}[f^{\lambda}(\phi^{\lambda}(\mathbf{x})) - y)]^2 + \lambda \text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi^{\lambda}(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi^{\lambda}(\mathbf{x}))) \lesssim \\
&\qquad \mathbb{E}_{\mathbf{x},y}[f_{\theta}(\phi(\mathbf{x})) - y)]]^2 + \lambda \text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x}))) + \varepsilon_{\text{stat}}.
\end{aligned}
$$

Now, we can use the above relationship, alongside bounding $(f_{\theta}(\mathbf{x}) - y)^2 \leq B_0$ (which holds since the function is bounded):

$$
\text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi^{\lambda}(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi^{\lambda}(\mathbf{x}))) \leq \text{disc}_{\mathcal{H}}(\mathbb{P}_{\pi}(\phi(\mathbf{x})), \mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x}))) + C_{\mathcal{H}} \cdot \zeta + \frac{2B_0}{\lambda}, \quad (24)
$$

where $\zeta \to 0$ as $\lambda \to 0$. Now, we can minimize the right hand side w.r.t. $\lambda$ and obtain the desired result:

$$
J(\pi) - J_{\theta}(\pi) \geq J(\mu_{\text{data}}) - J_{\theta}(\mu_{\text{data}}) - (2C + 1) \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x})), \mathbb{P}_{\pi}(\phi(\mathbf{x}))) - 2\varepsilon'_{\mathcal{F},\Phi} - \varepsilon_{\text{stat}},
$$

where $\varepsilon'_{\mathcal{F},\Phi} = \varepsilon_{\mathcal{F},\Phi} + h(B_0, C_{\mathcal{H}})$, where $h(\cdot)$ is a bias that is a property of the loss functions, the function classes $\mathcal{F}$ and $\Phi$, and the discrepancy metric. It does not depend on the learned functions $f_{\theta}$, $\phi$ learned or the distribution $\pi$. This completes the proof. □

*Remark* A.6 (**The interplay between $\lambda_0$, $\mathcal{F}$ and $\Phi$.**). In the proof above, we appeal to the continuity assumption (Assumption A.4) to enable us to bound the discrepancy measured in terms of the representation $\phi^*$ with respect to $\phi$, and this requires adjusting $\lambda$. This is unavoidable in the worst case if the function classes $\mathcal{F}$ and $\Phi$ are arbitrary and do not contain any solution $\phi^*$ that attains a small training error, while being somewhat invariant. In such cases, the only way to attain a lower-bound is to not be invariant, and this is what our bound would suggest by setting $\lambda$ to be on the smaller end of the spectrum. On the other hand, if the function class is more expressive, such that there exists a tuple $(f^*, \phi^*)$ that minimizes $\mathcal{L}_1(f, \phi))$ while being invariant, i.e., $\mathcal{L}_2(f^*, \phi^*)$ is small, then we can loosen this restriction over $\lambda$.

### A.1.3 Proof of Proposition 4.3

**Proposition A.7** (Performance guarantee for IOM)**.** *Under Assumption 4.1, the expected value of the ground truth objective under $\mu_{\text{OPT}}$, $J(\mu_{\text{OPT}})$ is lower bounded by:*

$$J(\mu_{\text{OPT}}) \gtrsim J(\mu_{data}) - \mathcal{O}\left(\sqrt{\frac{\log \frac{|\mathcal{F}||\Phi||\Pi|}{\delta}}{|\mathcal{D}|}} + \frac{\log \frac{|\mathcal{F}||\Phi||\Pi|}{\delta}}{|\mathcal{D}|}\right) + \underbrace{J_\theta(\mu_{\text{OPT}}) - J_\theta(\mu_{data})}_{(\circ)} - (*) - \varepsilon'_{\mathcal{F},\Phi}.$$

We restate the proposition above. There is a typo in the main paper, where the term $\varepsilon'_{\mathcal{F},\Phi}$ got deleted from the bound when moving from Proposition 4.2 to 4.3. The above restatement of Proposition 4.3 provides the complete statement.

*Proof.* To prove this proposition, we can utilize the result from Proposition A.5, but now we need to reason about a specific $\pi = \mu_{\text{OPT}}$, where $\mu_{\text{OPT}}$ depends on the learned $f_\theta$ and $\phi$. To account for such a $\pi$, we can instead bound the improvement $J(\pi) - J_\theta(\pi)$ for the worst possible $\pi \in \Pi$. While this does not change any term from the bound in Proposition A.5 that depends on the properties of the function class $\mathcal{F}$ or the representation class $\Phi$, this would still affect the statistical error, which would now require a uniform concentration argument over the policy class $\Pi$. Therefore, the bound from Proposition A.5 becomes:

$$J(\mu_{\text{OPT}}) - J_\theta(\mu_{\text{OPT}}) \geq J(\mu_{\text{data}}) - J_\theta(\mu_{\text{data}}) - C' \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{data}}}(\phi(\mathbf{x})), \mathbb{P}_{\mu_{\text{OPT}}}(\phi(\mathbf{x}))) - 2\varepsilon'_{\mathcal{F},\Phi} - \varepsilon_{\text{stat}}, \tag{25}$$

where $\varepsilon_{\text{stat}}$ is given by:

$$\varepsilon_{\text{stat}} = \mathcal{O}\left(\sqrt{\frac{\log\left(\frac{|\mathcal{F}||\Phi||\Pi|c_0}{\delta}\right)}{|\mathcal{D}|}}\right). \tag{26}$$

The term $\varepsilon_{\text{stat}}^2$ in the statement is a higher order term and is always dominated by $\varepsilon_{\text{stat}}$. Rearranging the terms completes the proof. □

### A.1.4 Extension to any re-weighting of the dataset

In this section, we will extend the theoretical result to compare the performace of the optimized design to the average objective value of any re-weighting of the dataset, and this by definition covers the comparison against the best design in the dataset. To do so, note that Proposition A.5 can be restated when comparing to $\mu_{\text{r}}$, a distribution induced by any specific re-weighting of the training dataset. This reformulation would give the following bound on the error:

$$J(\pi) - J_\theta(\pi) \geq \underbrace{J(\mu_{\text{r}}) - J_\theta(\mu_{\text{r}})}_{(\blacksquare)} - \underbrace{C_{\mathcal{F}} \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{r}}}(\phi(\mathbf{x})), \mathbb{P}_\pi(\phi(\mathbf{x})))}_{(*)} - 2\varepsilon'_{\mathcal{F},\Phi} - \varepsilon_{\text{stat}}. \tag{27}$$

Since, the choice of $\mu_{\text{r}}$ is arbitrary (i.e., it can be any arbitrary weighting of the dataset distribution $\mu_{\text{data}}$), we can now basically compute the tighest lower bound by maximizing the RHS with respect to the choice of $\mu_{\text{r}}$. This would amount to:

$$J(\pi) - J_\theta(\pi) \geq \max_{\mu_{\text{r}} \in T} \left[\underbrace{J(\mu_{\text{r}}) - J_\theta(\mu_{\text{r}})}_{(\blacksquare)} - \underbrace{C_{\mathcal{F}} \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{r}}}(\phi(\mathbf{x})), \mathbb{P}_\pi(\phi(\mathbf{x})))}_{(*)}\right] - 2\varepsilon'_{\mathcal{F},\Phi} - \varepsilon_{\text{stat}}, \tag{28}$$

where $T$ denotes the set of distributions that arising from all possible re-weightings of the training data. For the special case when $\mu_{\text{r}}$ simply corresponds to a Dirac-delta distribution centered at the best point in the training dataset, which we will denote as $\mathbf{x}_{\text{best}}$, and $\mu_{\text{r}} = \delta_{\mathbf{x}=\mathbf{x}_{\text{best}}}$, we can write down the bound as:

$$J(\mu_{\text{OPT}}) \gtrsim f(\mathbf{x}_{\text{best}}) + \underbrace{J_\theta(\pi) - \widehat{f}_\theta(\mathbf{x}_{\text{best}})}_{\text{improvement}} - \underbrace{C_{\mathcal{F}} \cdot \text{disc}_{\mathcal{H}}(\mathbb{P}_{\mu_{\text{r}}}(\phi(\mathbf{x})), \mathbb{P}_\pi(\phi(\mathbf{x})))}_{(*)} - 2\varepsilon'_{\mathcal{F},\Phi} - \varepsilon_{\text{stat}}. \tag{29}$$

When converting Equation 29 to a performance guarantee, note that $\varepsilon_{\text{stat}}$, which is used to bound the discrepancy $\text{disc}_{\mathcal{H}}$ in Equation 29 would not decay as the size of the dataset $|\mathcal{D}|$ increases, as our reference point is just one single point $\mathbf{x}_{\text{best}}$ in the dataset. That said, note that as the dataset size increases, the groundtruth value of $\mathbf{x}_{\text{best}}$ shall also increase which would tighten the RHS of Equation 29 as the dataset size increases.

Therefore, our bound presents a tradeoff: when comparing $J(\mu_{\text{OPT}})$ to the average value of the dataset $J(\mu_{\text{data}})$, we can attain favorable guarantees that guarantee improvement as the dataset size increases because the statistical error reduces, and on the other hand, when we compare $J(\mu_{\text{OPT}})$ to $f(\mathbf{x}_{\text{best}})$, the statistical error may not decay as the dataset size increases, however, the function value $f(\mathbf{x}_{\text{best}})$ will increase. In summary, our bound in either case is expected to improve as the dataset size increases.

**Remark:** Proposition 4.3 is inspired from theoretical results in offline reinforcement learning, where it is common to lower bound the improvement of the learned policy over the behavior policy (see for example, works on "safe policy improvment" [30, 41, 28, 53, 10]). When comparing to the best point in the dataset, or equivalently, the best in-support policy in the case of reinforcement learning, the bounds would inevitably become weaker, since it is not clear if *any* offline MBO or offline RL algorithm would improve over the best point in the dataset, in the worst-case problem instance. We have added this as a limitation of the theoretical results, but would remark that this limitation is also existent in other areas where one must learn to optimize decisions from offline data. Nevertheless, our empirical results do show that IOM does improve over the best point in the dataset, as shown in Table 2.

## A.2 Experiment Details

### A.2.1 Tasks and Datasets

**Tasks.** We provide brief description of the tasks we use for our empirical evaluation and the corresponding evaluation protocol below. More details can be found be found at Trabucco et al. [45].

- The Superconductor task aims at optimizing the superconducting materials which achieves high-critical temperatures. The original features are vectors containing specific physical properties, such as the mean atomic mass of the elements. Following prior work [45], we utilize the invertible input specification by representing different superconductors via a serialization of the chemical formula. The objective $y$ is the critical temperature the resulting material could achieve. Here, we have $\mathbf{x} \in \mathbb{R}^{86}$ and $y \in \mathbb{R}^{+}$. The original dataset in Design-Bench has 21263 samples in total. However, following prior work [46, 45, 51], we remove the top $20\%$ from the dataset to increase the difficulty and ensure the top-performing points are not visible during learning stage. **Evaluation**: To evaluate the performance of any given design, we use the oracle function provided in design-bench, which trains a random forest model (with the default hyper-parameters provided in [17]) on the whole original dataset.
- The Ant and D'Kitty Morphology task aim at designing the morphology of a quadrupedal robot such that the robot could be crawl quickly in certain direction. The Ant/D'Kitty task corresponds to the Ant/D'Kitty robot that used as the agent. Specifically, the input is the morphology of the agent with the objective value is the average return of the agent that uses the optimal control tailored for the morphology. More details about the controller design could be found at Trabucco et al. [45]. For the dimension of the design/feature space, we have $\mathbf{x} \in \mathbb{R}^{60}$ and $\mathbf{x} \in \mathbb{R}^{56}$ for D'Kitty. The original dataset contains 25009 samples for both Ant and D'Kitty. Following prior work [46, 51], we remove the $40\%$ of top-performing samples and use the rest as the new dataset. **Evaluation**: To evaluate any given morphology, it is passed into the MuJoCo Ant or D'Kitty environment, and a pre-trained policy is used (trained using Soft Actor-Critic for 10 million steps), the sum of rewards in a trajectory is used as the final score.
- The Hopper-Controller task aims at designing a set of weights of a particular neural network policy that achieves high return for the MuJoCo Hopper-v2 environment. Basically, the policy architecture is a three-layer neural network with 64 hidden units and 5126 weights in total with the algorithm is trained using Proximal Policy Optimization [40] (with default parameters provided in Hill et al. [19]). Specifically, the input is a vector of flattened weight tensors, with $\mathbf{x} \in \mathbb{R}^{5126}$, the objective is the expected return from a single roll-out using this

specific policy, with $y \in \mathbb{R}$. In the original dataset from Design-Bench, we have 3200 data points, and we do not discard any of them due to limited size of the dataset. **Evaluation:** To evaluate the design of a specific set of weight, which deploy PPO policy with this set of weight, and the sum of rewards of 1000 steps are collected using the agent, as the final score.

**Dataset.** For each task, we take the training dataset from design-bench, and in addition, randomly split it into training and validation with the ratio 7:3. The information of the original dataset is given in Table 1.

Table 1: Overview of the dataset information

| Dataset | Sample Size | Dimensions | Type of design space |
|---|---|---|---|
| **Superconductor** | 17010 | 86 | Continuous |
| **Ant Morphology** | 10004 | 60 | Continuous |
| **D'Kitty Morphology** | 10004 | 56 | Continuous |
| **Hopper Controller** | 3200 | 5126 | Continuous |
| **ChEMBL** | 1093 | 31 | Discrete |

### A.2.2 Baseline Descriptions

In this section, we briefly introduce the baseline methods for model-based optimization in an algorithmic level, all the implementation details for the baselines are referred to Trabucco et al. [45].

- **CbAS** [8] trains a density model via variational auto-encoder (VAE). Basically at each step, it updates the density model via a weighted ELBO objective which move towards better designs, which estimated by a pre-trained proxy model $\hat{f}(\mathbf{x})$; then generating new samples from the updated distributions to serve the new dataset, and repeated this process.
- **Anto.CbAS** [11] is an improved version of the CbAS. For CbAS, it uses the fixed pre-trained proxy model $\hat{f}(\mathbf{x})$ all the time, to guide the distribution towards the optimal one. However, the proxy model will suffer from distributional shift as the density model changes. Hence, Auto.CbAS corrects the distributional shift by re-training the proxy model under the new optimized density model, via importance sample weights.
- **MINs** [26] learn an inverse mapping $f^{-1}$ from the objective value $y$ to the corresponding design $\mathbf{x}$. The inverse map is trained using the conditional GAN. Then it tries to search for the optimal $y$ value during the optimization, and the optimized design is found by sampling from this inverse map.
- **COMs** [46] learns a conservative proxy model $\hat{f}(\mathbf{x})$. Compared with the vanilla gradient ascent, COMs augments the regression objective with an additional regularization term which aims to penalize the over-estimation error of the out-of-distribution points. The final optimal designed point is found by performing gradient ascent on this learned proxy model.
- **RoMA** [51] tries to overcome the brittleness of the deep neural networks that used in the proxy model. Basically, the method is a two-stage process, with the first stage is to train a proxy model $\hat{f}$ with Gaussian smoothing of the inputs, and the second stage is to update the solution while simultaneously adapt the model to achieve the smoothness in the input level for the current optimized point.
- **BO-qEI** [49] refers the Bayesian Optimization with quasi-Expected Improvement acquisition function. The method tries to fit the proxy model $\hat{f}$, and maximizing it via Gaussian process.
- **CMA-ES** [18] refers the covariance matrix adaptation. The idea is to maintain a Gaussian distribution of the optimized design. At each step, samples of designs are drawing from the Gaussian distribution, and the covariance matrix is re-fitted by the top-performing designs estimated by a pre-trained proxy model $\hat{f}(\mathbf{x})$.
- **Grad./Grad.Min/Grad.Mean** [46] are a family of methods that use gradient ascent to find the optimal design. All of them require a pre-trained proxy model. For Grad., the proxy model is trained by minimizing the $L_2$ norm of $\hat{f}(\mathbf{x})$ and the ground-truth value $y$. For Grad.Min, it learns 5 different proxy models, and use the minimum of it as the predicted

value. For Grad.Mean, it uses the mean of the 5 models. In the optimization stage, gradient ascent on the proxy model is used to find the optimized point.

- **REINFORCE** [48] pre-trains a proxy model $\hat{f}(\mathbf{x})$ and use the policy gradient method to find the optimal design distributions that maximizes the $\hat{f}(\mathbf{x})$.

#### A.2.3 Implementation Details

For all the baseline methods, we directly report performance numbers associated with DesignBench [45]. Below, we now discuss implementation details for the method developed in this paper, IOM. Our implementation builds off of the official implementation of COMs, provided by the authors Trabucco et al. [46]

**Data preprocessing.** Following prior work [46], we perform standardization for both the input $\mathbf{x}$ and objective value $y$ before training the objective model $\widehat{f}$. Essentially, we standardize $\mathbf{x}$ to $\tilde{\mathbf{x}} := \frac{\mathbf{x}-\mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}$, with $\mu_{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ stands for the empirical mean and variance for $\mathbf{x}$, similarly for $\tilde{y}$ with $\tilde{y} := \frac{y-\mu_y}{\sigma_y}$.

**Neural networks representing $\widehat{f}$ and $\phi$.** We parameterize the learned model $\widehat{f}$ and the representation $\phi(\cdot) \in \mathbb{R}^{64}$ using three hidden layer neural networks, that assume identical architectures for all the tasks. The representation network, $\phi_\eta(\mathbf{x})$ is a neural network with two hidden layers of size 2048, followed by Leaky ReLU (using default leak value of 0.3). The representation is of size 128. In some preliminary experiments, we observed that our method, IOM was pretty robust to the dimension of the representation, so we simply chose to utilize 128-dimensional representations for all the task, with no tuning. For the network representing $\widehat{f}$, we utilize a neural network with two hidden layers of size 1024, followed by Leaky ReLU (using default leak 0.3), the output is a 1-dimensional scalar used to predict the value $y$.

**Hyperparameters.** We utilize identical hyperparameters for all tasks, and these hyperparameters are taken directly from the implementation of Trabucco et al. [46]. For training the learned model $f_\theta(\phi(\cdot))$, we use Adam optimizer with default learning rate 0.001 and $(\beta_1, \beta_2) = (0.9, 0.999)$. The batch size is 128 and the number of training epochs is 300. For training the representation network, the optimizer is Adam with learning rate 0.0003 and default $(\beta_1, \beta_2) = (0.9, 0.999)$. We represent $\mu_{\text{OPT}}$ as a non-parameteric distribution represented by 128 particles, initialized from 128 randomly sampled inputs $\mathbf{x}$ in the dataset. We then alternate between performing one gradient step on $\mu_{\text{OPT}}$ and one gradient step on $f_\theta$ and $\phi$. To select a value for the hyperparameter $\lambda$ (weight for the invariance regularizer term), we run our method with a wide range of $\lambda$ values $\Lambda = \{0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 100.0\}$. In the main experimental results we report, following the convention in prior work [45], we choose $\lambda$ to be uniform across all tasks, but $\lambda$ is chosen via online evaluation. For our tuning procedure, we apply our tuning procedure where we keep the top 45% models which achieve a discriminator loss closes to 0.25, and select the one with the smallest validation prediction error among it . We perform an ablation of this strategy and the quantile of models that are kept in Section 6.2.

For the IOM-C variant we show in our experiments, we set the initial value of the Lagrange multiplier $\alpha = 0.3$–and this value is taken directly from Trabucco et al. [46]–and update it using Adam optimizer with learning rate 0.01.

We release the code for our method along with additional details at the following anonymous website: https://sites.google.com/view/iom-neurips.

### A.3 Additional Results

In this section, we provide additional experimental results.

#### A.3.1 Overall Performance Comparisons

In Table 2, we provide the comparison of IOM and IOM-C with all baselines (including RoMA). To understand the trend in aggregated results, please check the plots in Figure 4.

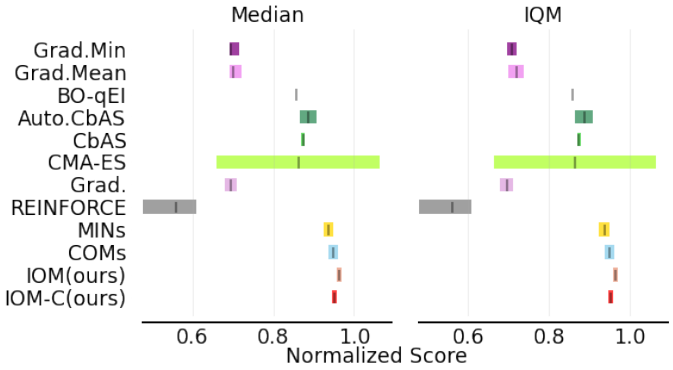


Figure 4: Median and IQM [1] (with 95% Stratified Bootstrap CIs) for the aggregated normalized score on Design-bench, for all the baseline methods. As we could not find the individual runs for RoMA [51], we report the mean and standard deviation for RoMA (copied from the results in Yu et al. [51]) in Appendix A.3.

Table 2: **Comparative evaluation of IOM and IOM-C against prior methods** in terms of the mean 100th-percentile score (following the protocol in Trabucco et al. [45]) and its standard deviation over 5 trials (the number for ROMA is copied from Yu et al. [51], and it is over 16 trails). We have now additionally evaluated the LCB+GP baseline that optimizes the design against the lower-confidence bound estimate obtained from the posterior of a GP fit to the training data. Observe that IOM outperforms this approach as well.

| | **Superconductor** | **Ant Morphology** | **D'Kitty Morphology** | **Hopper Controller** |
|---|---|---|---|---|
| $\mathcal{D}$ (**best**) | 0.399 | 0.565 | 0.884 | 1.0 |
| Auto. CbAS | 0.421 $\pm$ 0.045 | 0.882 $\pm$ 0.045 | 0.906 $\pm$ 0.006 | 0.137 $\pm$ 0.005 |
| CbAS | **0.503** $\pm$ **0.069** | 0.876 $\pm$ 0.031 | 0.892 $\pm$ 0.008 | 0.141 $\pm$ 0.012 |
| MINs | 0.469 $\pm$ 0.023 | 0.913 $\pm$ 0.036 | 0.945 $\pm$ 0.012 | 0.424 $\pm$ 0.166 |
| COMs | 0.439 $\pm$ 0.033 | 0.944 $\pm$ 0.016 | 0.949 $\pm$ 0.015 | **2.056** $\pm$ **0.314** |
| RoMA | **0.562** $\pm$ **0.030** | 0.875 $\pm$ 0.013 | **1.036** $\pm$ **0.042** | 1.867 $\pm$ 0.282 |
| BO-qEI | 0.402 $\pm$ 0.034 | 0.819 $\pm$ 0.000 | 0.896 $\pm$ 0.000 | 0.550 $\pm$ 0.118 |
| CMA-ES | 0.465 $\pm$ 0.024 | **1.214** $\pm$ **0.732** | 0.724 $\pm$ 0.001 | 0.604 $\pm$ 0.215 |
| Grad. | **0.518** $\pm$ **0.024** | 0.293 $\pm$ 0.023 | 0.874 $\pm$ 0.022 | 1.035 $\pm$ 0.482 |
| Grad. Min | 0.506 $\pm$ 0.009 | 0.479 $\pm$ 0.064 | 0.889 $\pm$ 0.011 | **1.391** $\pm$ **0.589** |
| Grad. Mean | 0.499 $\pm$ 0.017 | 0.445 $\pm$ 0.080 | 0.892 $\pm$ 0.011 | **1.586** $\pm$ **0.454** |
| REINFORCE | 0.481 $\pm$ 0.013 | 0.266 $\pm$ 0.032 | 0.562 $\pm$ 0.196 | -0.020 $\pm$ 0.067 |
| LCB+GP | N/A | 0.765 $\pm$ 0.056 | 0.871 $\pm$ 0.009 | 1.736 $\pm$ 0.261 |
| **IOM (Ours)** | **0.504** $\pm$ **0.040** | 0.977 $\pm$ 0.009 | 0.949 $\pm$ 0.010 | **2.444** $\pm$ **0.080** |
| **IOM-C (Ours)** | **0.511** $\pm$ **0.037** | 0.966 $\pm$ 0.012 | 0.940 $\pm$ 0.007 | **1.482** $\pm$ **0.650** |

### A.4 Additional results for early stopping the optimization of $\mu_{\text{OPT}}$

We provide additional results for the effectiveness of our tuning strategy for early stopping. Results for the additional two tasks are presented in Figure 5. The top row shows the objective value of the optimized point as a function of the number of training epochs, and the bottom row shows the prediction error on the validation set. We use the vertical line to denote the point with the smallest MSE by our early stopping strategy. Among all tasks, our strategy is able to find a good checkpoint and this verifies the efficacy of our approach on different domains.

### A.5 Performance of IOM and IOM-C under different invariance regularizer $\lambda$

In this section, we provide the final performance of IOM and IOM-C, under different values of the hyperparameter $\lambda$ used for training (Equation 4). Results for all tasks are displayed in Figure 7. As is the case with any other prior method for data-driven decision making that adds a regularizer, both IOM and IOM-C are also sensitive to the hyper-parameter $\lambda$, which demonstrates the necessity of our proposed tuning scheme. And fortunately, as observed in Section 6, our tuning scheme can

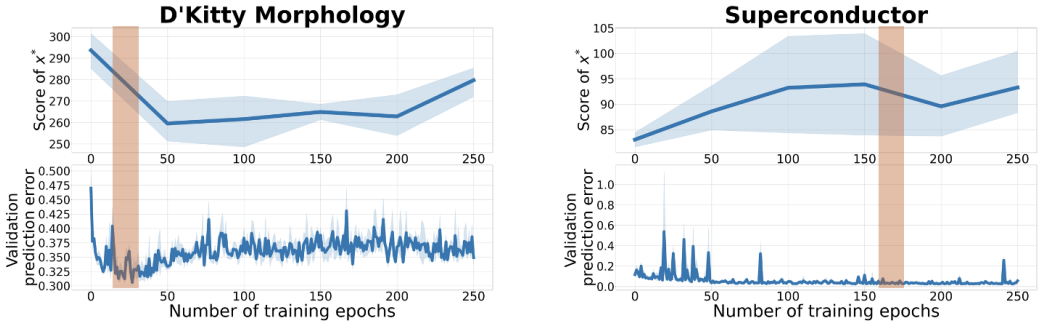


Figure 5: **Visualizing our early stopping scheme on two more tasks.** As per Section 5, the checkpoint selected is the epoch that has the smallest validation prediction error, and early stopping at this point generally performs well.

indeed attain good performance, for many choices of the $\epsilon$ (see Figure 3), making it easier to tune this hyperparameter $\lambda$ in practice.

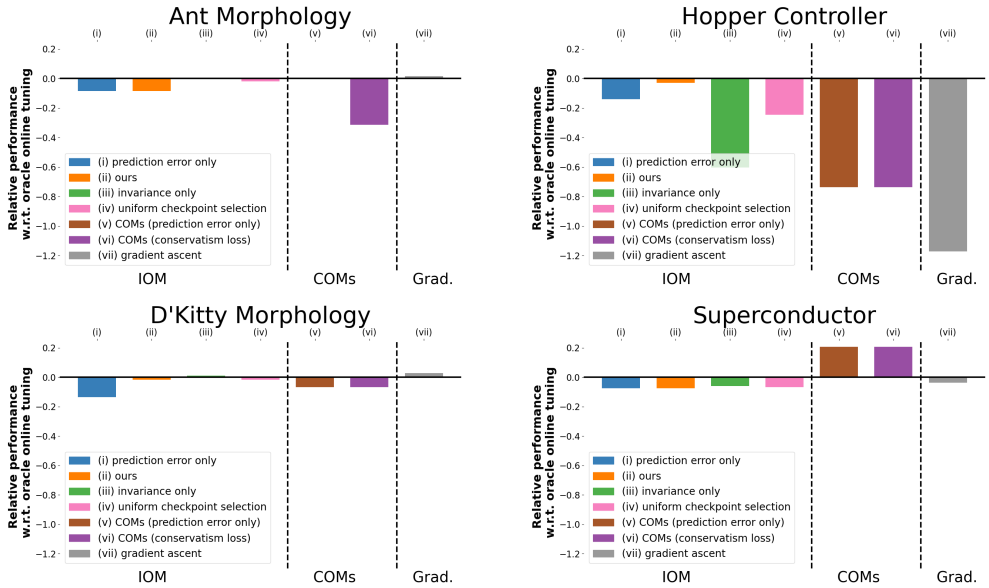


Figure 6: Suboptimality gaps of various offline tuning strategies for IOM when compared in each individual task.

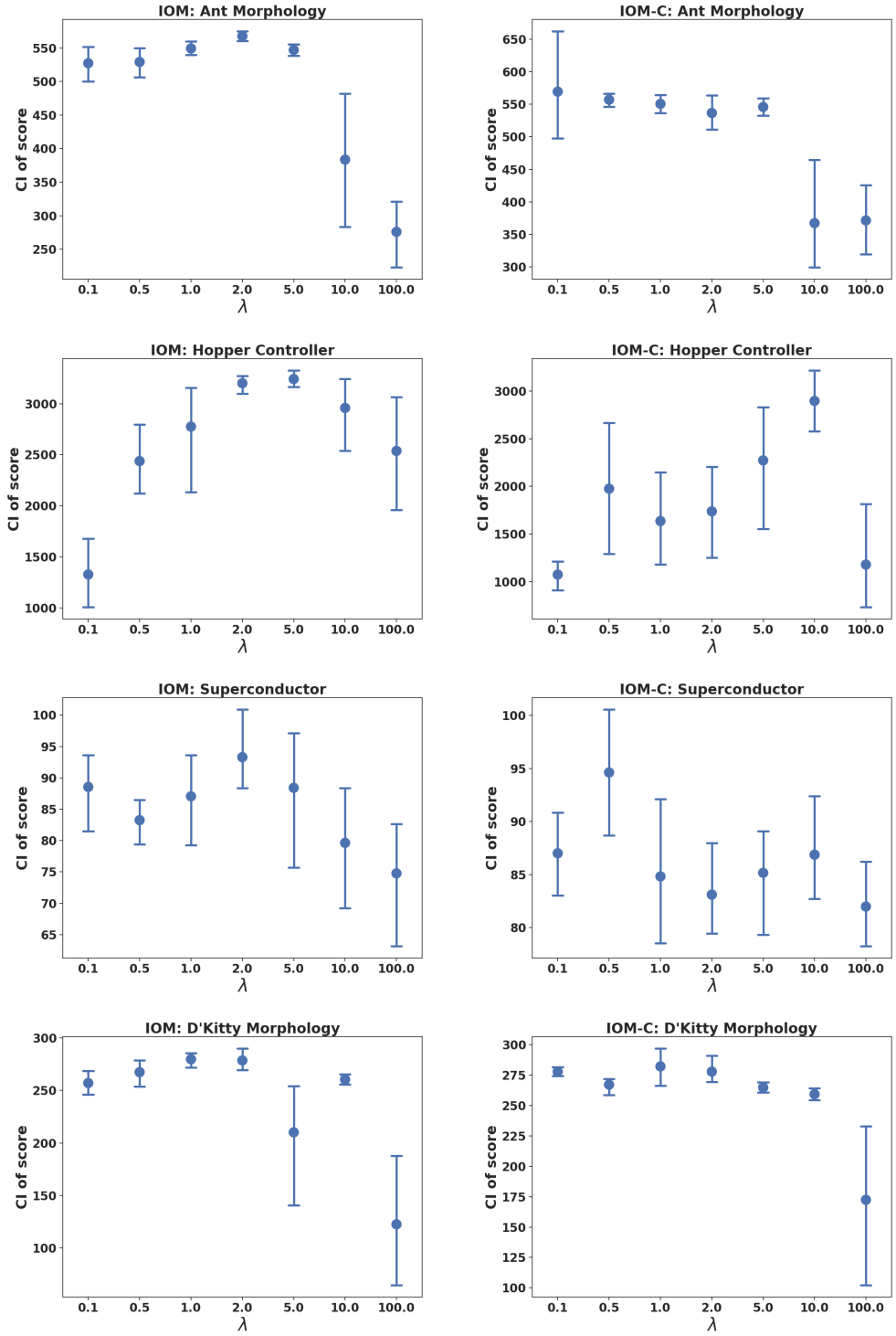


Figure 7: Score of final optimized $\mathbf{x}^*$ (with 2 standard deviations) of IOM and IOM-C under different invariance regularizer $\lambda$

### A.6 Additional Results

**Computational cost of tuning IOM via our offline workflow.** In order to tune IOM via our offline workflow we first run IOM with multiple different hyperparameters, concretely, in our experiments, we use 7 different hyperparameters, and then apply our offline tuning strategy, which only utilizes the loss values of the various runs, without incurring any significant computational cost of its own.

Thus, our hyperparameter tuning strategy does not incur additional computational cost. When it comes to the cost of running IOM, we expect that beyond implementation differences, the cost of running IOM should not be significantly worse than COMs: the only additional component trained by IOM is the discriminator that is used to compute the $\chi^2$-divergence on the representation, which in our implementation only adds 2 more fully connected layers.

**COMs lead to worse training prediction error and more non-smooth functions compared to IOM.** Next we attempt to gain insights into what might explain why IOM outperforms COMs in several of our tasks. We suspect that this is because applying a conservatism regularizer on the learned function value itself, like COMs [46], does in fact distort the learned objective function more than IOM. To understand if this is the case empirically, we plot two quantities: **(a)** the value of the training prediction error: $\mathbb{E}_{\mathbf{x},y\sim\mathcal{D}}\left[(\widehat{f}_\theta(\mathbf{x})-y)^2\right]$ for the best hyperparameter for COMs and IOM in Figure 8, and **(b)** the "smoothness" of the learned function w.r.t. input, measured via the norm of the gradient of $f_\theta$ with respect to $\mathbf{x}$, i.e., $\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[||\nabla_x f_\theta(\phi(\mathbf{x})||_2^2\right]$ in Figure 9. Observe that in both the Ant Morphology and Hopper Controller tasks, IOM is able to minimize the prediction error better than COMs, and the gradient norm for COMs is strictly higher than IOM indicating that conservatism on learned function leads to more non-smooth functions and reduces the ability of the learned function to fit the offline dataset.

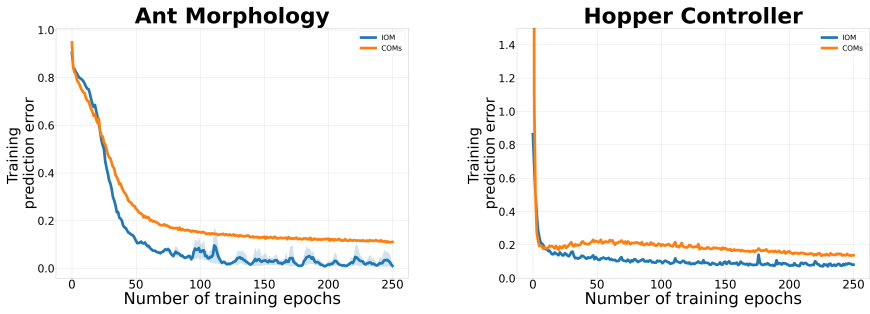


Figure 8: **Comparison of prediction error between IOM and COMs.** Note that COMs attains a worse prediction error than IOM.

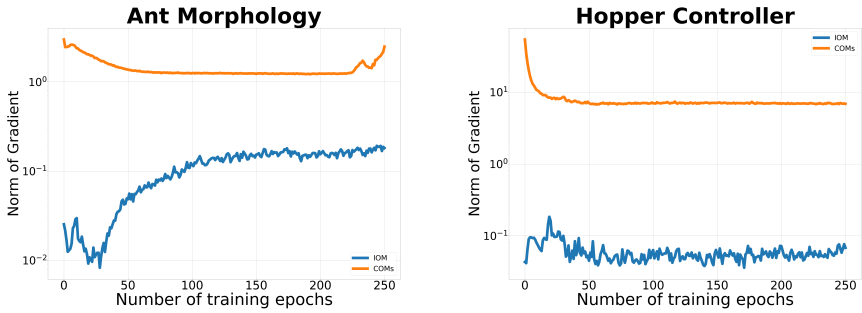


Figure 9: **Comparison of gradient norm between IOM and COMs during training (log scale).** Note that the gradient norm for COMs is clearly higher, indicating that COMs learn a more non-smooth function than IOM.

## A.7 Intuition Behind How IOM Works

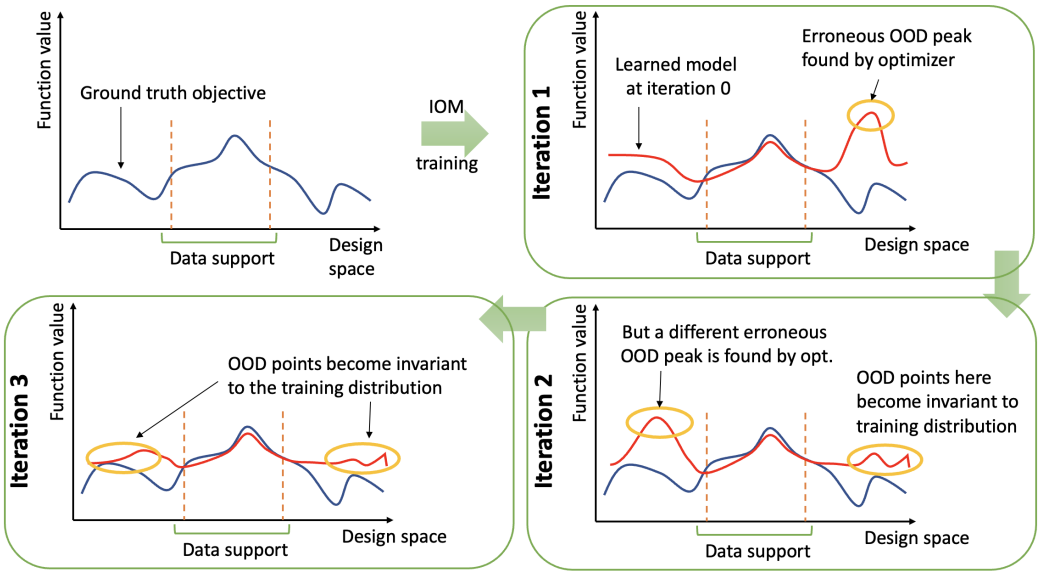


Figure 10: **Figure illustrating the intuition behind IOM.** When we apply the invariance regularizer prescribed by IOM during training, this invariance regularizer would attempt to make the distribution of the representation of the out-of-distribution (OOD) designs found by the optimizer match the distribution of the representations in the training dataset, which would make these points take on a function value that matches the average dataset value. By running this training multiple times, the IOM training procedure would gradually suppress the value of all out-of-distribution erroneous peaks in the learned function, and will enable us to find an optimal design that lies within the training distribution (though not necessarily in the training dataset.)