Bidirectional Learning for Offline Infinite-width Model-based Optimization

Can(Sam) Chen[1], Yingxue Zhang[2], Jie Fu[3], Xue Liu[1], Mark Coates[1]

[1]McGill University, [2] Huawei Noah's Ark Lab, [3] Beijing Academy of Artificial Intelligence
can.chen@mail.mcgill.ca, yingxue.zhang@huawei.com,
fujie@baai.ac.cn, xueliu@cs.mcgill.ca, mark.coates@mcgill.ca

Abstract

In offline model-based optimization, we strive to maximize a black-box objective function by only leveraging a static dataset of designs and their scores. This problem setting arises in numerous fields including the design of materials, robots, DNA sequences, and proteins. Recent approaches train a deep neural network (DNN) on the static dataset to act as a proxy function, and then perform gradient ascent on the existing designs to obtain potentially high-scoring designs. This methodology frequently suffers from the out-of-distribution problem where the proxy function often returns poor designs. To mitigate this problem, we propose ***B**i**D**irectional learning for offline **I**nfinite-width model-based optimization* (**BDI**). BDI consists of two mappings: the forward mapping leverages the static dataset to predict the scores of the high-scoring designs, and the backward mapping leverages the high-scoring designs to predict the scores of the static dataset. The backward mapping, neglected in previous work, can distill more information from the static dataset into the high-scoring designs, which effectively mitigates the out-of-distribution problem. For a finite-width DNN model, the loss function of the backward mapping is intractable and only has an approximate form, which leads to a significant deterioration of the design quality. We thus adopt an infinite-width DNN model, and propose to employ the corresponding neural tangent kernel to yield a closed-form loss for more accurate design updates. Experiments on various tasks verify the effectiveness of BDI. The code is available here.

1 Introduction

Designing a new object or entity with desired properties is a fundamental problem in science and engineering [1]. From material design [2] to protein design [3, 4], many works rely on interactive access with the unknown objective function to propose new designs. However, in real-world scenarios, evaluation of the objective function is often expensive or dangerous [2–6], and thus it is more reasonable to assume that we only have access to a static (offline) dataset of designs and their scores. This setting is called offline model-based optimization. We aim to find a design to maximize the unknown objective function by only leveraging the static dataset.

A common approach to solve this problem is to train a deep neural network (DNN) model on the static dataset to act as a proxy function parameterized as $f_{\boldsymbol{\theta}}(\cdot)$. Then the new designs are obtained by performing gradient ascent on the existing designs with respect to $f_{\boldsymbol{\theta}}(\cdot)$. This approach is attractive because it can leverage the gradient information of the DNN model to obtain improved designs. Unfortunately, the trained DNN model is only valid near the training distribution and performs poorly outside of the distribution. More specifically, the designs generated by directly optimizing $f_{\boldsymbol{\theta}}(\cdot)$ are scored with erroneously high values [7, 8]. As shown in Figure 1, the proxy function trained on the

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

static dataset $p_{1,2,3}$ overestimates the ground truth objective function, and the seemingly high-scoring design p_{grad} obtained by gradient ascent has a low ground truth score p'_{grad}.

Recent works [7–9] address the out-of-distribution problem by imposing effective inductive biases on the DNN model. The method in [8] learns a proxy function $f_{\boldsymbol{\theta}}(\cdot)$ that lower bounds the ground truth scores on out-of-distribution designs. The approach in [9] bounds the distance between the proxy function and the objective function by normalized maximum likelihood. In [7], Yu et al. use a local smoothness prior to overcome the brittleness of the proxy function and thus avoid poor designs.

As demonstrated in Figure 1, these previous works try to better fit the proxy function to the ground truth function from a *model* perspective, and then obtain high-scoring designs by gradient ascent regarding the proxy function. In this paper, we investigate a method that does not focus on regularizing the model, but instead ensures that the proposed designs can be used to predict the available *data*. As illustrated in Figure 1, by ensuring the high-scoring design p_{ours} can predict the scores of the static dataset $p_{1,2,3}$ (backward) and vice versa (forward), p_{ours} distills more information from $p_{1,2,3}$, which makes p_{ours} more aligned with $p_{1,2,3}$, leading to p'_{ours} with a high ground truth score.

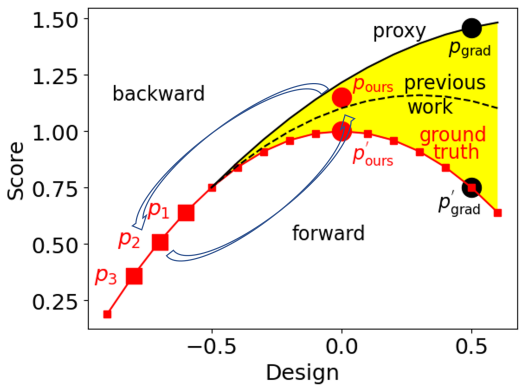

Figure 1: Illustration of motivation.

We propose ***Bi**directional learning for offline **I**nfinite-width model-based optimization* (**BDI**) between the high-scoring designs and the static dataset (a.k.a. low-scoring designs). BDI has a forward mapping from low-scoring to high-scoring and a backward mapping from high-scoring to low-scoring. The backward mapping means the DNN model trained on the high-scoring designs is expected to predict the scores of the static dataset, and vice versa for the forward mapping. To compute the prediction losses, we need to know the scores of the high-scoring designs. Inspired by [10], where a predefined reward (score) is used for reinforcement learning, we set a predefined target score for the high-scoring designs. By minimizing the prediction losses, the bidirectional mapping distills the knowledge of the static dataset into the high-scoring designs. This ensures that the high-scoring designs are more aligned with the static dataset and thus effectively mitigates the out-of-distribution problem. For a finite-width DNN model, the loss function of the backward mapping is intractable and can only yield an approximate form, which leads to a significant deterioration of the design quality. We thus adopt a DNN model with infinite width, and propose to adopt the corresponding neural tangent kernel (NTK) [11, 12] to produce a closed-form loss function for more accurate design updates. This is discussed in Section 4.5. Experiments on multiple tasks in the design-bench [1] verify the effectiveness of BDI.

To summarize, our contributions are three-fold:

- We propose bidirectional learning between the high-scoring designs and the static dataset, which effectively mitigates the out-of-distribution problem.
- To enable a closed-form loss function, we adopt an infinite-width DNN model and introduce NTK into the bidirectional learning, which leads to better designs.
- We achieve state-of-the-art results on various tasks, which demonstrates the effectiveness of BDI.

2 Preliminaries

2.1 Offline Model-based Optimization

Offline model-based optimization is formally defined as:

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} f(\boldsymbol{x}), \tag{1}$$

where the objective function $f(\boldsymbol{x})$ is unknown, but we have access to a size N dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1)\}, \cdots, \{(\boldsymbol{x}_{\text{N}}, y_{\text{N}})\}$, where $\boldsymbol{x}_i$ represents a certain design and y_i denotes the design score.

The design could, for example, be a drug, an aircraft or a robot morphology. Denote the feature dimension of a design $\boldsymbol{x}$ as D and then the static dataset $\mathcal{D}$ can also have a vector representation $(\boldsymbol{X}_l, \boldsymbol{y}_l)$ where $\boldsymbol{X}_l \in \mathcal{R}^{\mathrm{N}\times\mathrm{D}}, \boldsymbol{y}_l \in \mathcal{R}^{\mathrm{N}}$.

A common approach is fitting a DNN model $f_{\boldsymbol{\theta}}(\cdot)$ with parameters $\boldsymbol{\theta}$ to the static dataset:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} (f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)^2 \,. \tag{2}$$

After that, the high-scoring design $\boldsymbol{x}^*$ can be obtained by optimizing $\boldsymbol{x}$ against the proxy function $f_{\boldsymbol{\theta}^*}(\boldsymbol{x})$ by gradient ascent steps:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \eta \nabla_{\boldsymbol{x}} f_{\boldsymbol{\theta}}(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_t} \,, \quad \text{for } t \in [0, \mathrm{T}-1] \,, \tag{3}$$

where T is the number of steps. The high-scoring design $\boldsymbol{x}^*$ can be obtained as $\boldsymbol{x}_{\mathrm{T}}$. This simple gradient ascent method suffers from the out-of-distribution problem — the proxy function $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is not accurate for unseen designs, and its actual score is usually considerably lower than predicted [8].

2.2 Infinitely Wide DNN and Neural Tangent Kernel

Deep infinitely wide neural network models have recently received substantial attention. Previous works establish the correspondence between infinitely wide neural networks and kernel methods [11–14]. In the infinite width assumption, wide neural networks trained by gradient descent with a squared loss yield a neural tangent kernel (NTK), which is specified by the network architecture. More specifically, given two samples (designs) $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the NTK $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ measures the similarity between the two samples from the perspective of the DNN model.

The NTKs have outperformed their finite network counterparts in various tasks and achieved state-of-the-art performance for classification tasks such as CIFAR10 [15]. The approach in [16] leverages the NTK to distill a large dataset into a small one to retain as much of its information as possible and the method in [17] uses the NTK to learn black-box generalization attacks against DNN models.

3 Method

In this section, we present ***B**i**D**irectional learning for offline **I**nfinite-width model-based optimization* (**BDI**). First, to mitigate the out-of-distribution problem, we introduce bidirectional learning between the high-scoring designs and the static dataset in Section 3.1. For a finite-width DNN model, the loss function is intractable. Approximate solutions lead to significantly poorer designs. We therefore propose the adoption of an infinite-width DNN model, and leverage its NTK to yield a closed-form loss and more accurate updates in Section 3.2.

3.1 Bidirectional Learning

We use $\boldsymbol{X}_h \in \mathcal{R}^{\mathrm{M}\times\mathrm{D}}$ to represent the high-scoring designs, where M represents the number of high-scoring designs. The high-scoring designs $\boldsymbol{X}_h$ are potentially outside of the static dataset distribution. To mitigate the out-of-distribution problem, we propose bidirectional learning between the designs and the static dataset, which leverages one to predict the other and vice versa. This can distill more information from the static dataset into the high-scoring designs. To compute the prediction losses, we need to know the scores of the high-scoring designs. Inspired by [10], in which a reward (score) is predefined for reinforcement learning, we set a predefined target score y_h for every high-scoring design. More precisely, we normalize the scores of the static dataset to have unit Gaussian statistics following [8] and then choose y_h to be larger than the maximal score in the static dataset. We set y_h as 10 across all tasks but demonstrate that BDI is robust to different choices in Section 4.6. Thus, the high-scoring designs can be written as $(\boldsymbol{X}_h, \boldsymbol{y}_h)$, and the goal is to find $\boldsymbol{X}_h$.

Forward mapping. Similar to traditional gradient ascent methods, the forward mapping leverages the static dataset to predict the scores of the high-scoring designs. To be specific, the DNN model $f^l_{\boldsymbol{\theta}}(\cdot)$ trained on $(\boldsymbol{X}_l, \boldsymbol{y}_l)$ is encouraged to predict the scores $\boldsymbol{y}_h$ given $\boldsymbol{X}_h$. Thus the forward loss function can be written as:

$$\mathcal{L}_{l2h}(\boldsymbol{X}_h) = \frac{1}{\mathrm{M}} \|\boldsymbol{y}_h - f^l_{\boldsymbol{\theta}^*}(\boldsymbol{X}_h)\|^2 \,, \tag{4}$$

where $\boldsymbol{\theta}^*$ is given by

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{\mathrm{N}} \|\boldsymbol{y}_l - f^l_{\boldsymbol{\theta}}(\boldsymbol{X}_l)\|^2 + \frac{\beta}{\mathrm{N}} \|\boldsymbol{\theta}\|^2 \,. \tag{5}$$

Then we can minimize $\mathcal{L}_{l2h}(\boldsymbol{X}_h)$ against $\boldsymbol{X}_h$ to update the high-scoring designs. $\mathcal{L}_{l2h}(\boldsymbol{X}_h)$ is similar to the gradient ascent process in Eq.(3); the only difference is that gradient ascent aims to gradually increase the score prediction whereas minimizing $\mathcal{L}_{l2h}(\boldsymbol{X}_h)$ pushes the score prediction towards the (high) predefined target score y_h.

Backward mapping. Similarly, the DNN model $f^h_{\boldsymbol{\theta}}(\cdot)$ trained on $(\boldsymbol{X}_h, \boldsymbol{y}_h)$ should be able to predict the scores $\boldsymbol{y}_l$ given $\boldsymbol{X}_l$. The backward loss function is

$$\mathcal{L}_{h2l}(\boldsymbol{X}_h) = \frac{1}{\mathrm{N}} \|\boldsymbol{y}_l - f^h_{\boldsymbol{\theta}^*(\boldsymbol{X}_h)}(\boldsymbol{X}_l)\|^2 \,, \tag{6}$$

where $\boldsymbol{\theta}^*(\boldsymbol{X}_h)$ is given by

$$\boldsymbol{\theta}^*(\boldsymbol{X}_h) = \arg\min_{\boldsymbol{\theta}} \frac{1}{\mathrm{M}} \|\boldsymbol{y}_h - f^h_{\boldsymbol{\theta}}(\boldsymbol{X}_h)\|^2 + \frac{\beta}{\mathrm{M}} \|\boldsymbol{\theta}\|^2 \,. \tag{7}$$

Overall loss. The forward mapping and the backward mapping together align the high-scoring designs with the static dataset, and the BDI loss can be compactly written as:

$$\mathcal{L}(\boldsymbol{X}_h) = \mathcal{L}_{l2h}(\boldsymbol{X}_h) + \mathcal{L}_{h2l}(\boldsymbol{X}_h) \,, \tag{8}$$

where we aim to optimize the high-scoring designs $\boldsymbol{X}_h$.

3.2 Closed-form Solver via Neural Tangent Kernel

For a finite-width DNN model $f^h_{\boldsymbol{\theta}}(\cdot)$, the high-scoring designs $\boldsymbol{X}_h$ of Eq.(6) only exist in $\boldsymbol{\theta}^*(\boldsymbol{X}_h)$ from Eq.(7), which is intractable and does not have a closed-form solution. The solution of Eq.(7) is often approximately obtained by a gradient descent step [18, 19],

$$\boldsymbol{\theta}^*(\boldsymbol{X}_h) = \boldsymbol{\theta} - \frac{\eta}{\mathrm{M}} \frac{\partial \|\boldsymbol{y}_h - f^h_{\boldsymbol{\theta}}(\boldsymbol{X}_h)\|^2}{\partial \boldsymbol{\theta}} \,, \tag{9}$$

where η denotes the learning rate. This approximate solution $\boldsymbol{\theta}^*(\boldsymbol{X}_h)$ leads to inaccurate updates to the designs $\boldsymbol{X}_h$ through Eq.(6) and leads to much poorer designs, especially for high-dimensional settings, as we illustrate in Section 4.5.

We thus adopt a DNN model $f_{\boldsymbol{\theta}}(\cdot)$ with infinite width, and propose to leverage the corresponding NTK to produce a closed-form $\boldsymbol{\theta}^*(\boldsymbol{X}_h)$ [11, 12] and then a closed-form loss function of Eq.(6).

Neural tangent kernel. Denote by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ the kernel function between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ introduced by the infinite-width DNN $f_{\boldsymbol{\theta}}(\cdot)$. We use $\boldsymbol{K}_{\boldsymbol{X}_l\boldsymbol{X}_l} \in \mathbf{R}^{\mathrm{N}\times\mathrm{N}}$, $\boldsymbol{K}_{\boldsymbol{X}_h\boldsymbol{X}_h} \in \mathbf{R}^{\mathrm{M}\times\mathrm{M}}$, $\boldsymbol{K}_{\boldsymbol{X}_l\boldsymbol{X}_h} \in \mathbf{R}^{\mathrm{N}\times\mathrm{M}}$ and $\boldsymbol{K}_{\boldsymbol{X}_h\boldsymbol{X}_l} \in \mathbf{R}^{\mathrm{M}\times\mathrm{N}}$ to represent the corresponding covariance matrices induced by the kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. By leveraging the closed-form $\boldsymbol{\theta}^*(\boldsymbol{X}_h)$ given by [11, 12], we can compute the closed-form $\mathcal{L}_{h2l}(\boldsymbol{X}_h)$ in Eq.(6) as:

$$\mathcal{L}_{h2l}(\boldsymbol{X}_h) = \frac{1}{\mathrm{N}} \|\boldsymbol{y}_l - \boldsymbol{K}_{\boldsymbol{X}_l\boldsymbol{X}_h}(\boldsymbol{K}_{\boldsymbol{X}_h\boldsymbol{X}_h} + \beta\boldsymbol{I})^{-1}\boldsymbol{y}_h\|^2 \,, \tag{10}$$

where $\beta{>}0$ is a fixed regularization parameter.

Similar to [20], to focus on higher-scoring designs in the static dataset, we assign a weight to every design based on its score and recompute the loss:

$$\mathcal{L}_{h2l}(\boldsymbol{X}_h) = \|\boldsymbol{\omega}_l \cdot (\boldsymbol{y}_l - \boldsymbol{K}_{\boldsymbol{X}_l\boldsymbol{X}_h}(\boldsymbol{K}_{\boldsymbol{X}_h\boldsymbol{X}_h} + \beta\boldsymbol{I})^{-1}\boldsymbol{y}_h)\|^2 \,, \tag{11}$$

where $\boldsymbol{\omega}_l = \sqrt{\mathrm{softmax}(\alpha\boldsymbol{y}_l)}$ represents the design weight vector and the weight parameter $\alpha{\geq}0$ is a constant. Similarly, we have the forward loss function,

$$\mathcal{L}_{l2h}(\boldsymbol{X}_h) = \|\boldsymbol{\omega}_h \cdot (\boldsymbol{y}_h - \boldsymbol{K}_{\boldsymbol{X}_h\boldsymbol{X}_l}(\boldsymbol{K}_{\boldsymbol{X}_l\boldsymbol{X}_l} + \beta\boldsymbol{I})^{-1}\boldsymbol{y}_l)\|^2 \,, \tag{12}$$

where $\boldsymbol{\omega}_h = \frac{\mathbf{1}}{\sqrt{M}}$ since every element in $\boldsymbol{y}_h$ is the same in our paper.

Besides the advantage of the closed-form computation, introducing the NTK into the bidirectional learning can also avoid the expensive high-order derivative computations brought by Eq.(9), which makes our BDI an efficient first-order optimization method.

Analysis of M=1. When we aim for one high-scoring design ($\mathrm{M} = 1$), Eq.(11) has a simpler form:

$$\begin{aligned}\mathcal{L}_{h2l}(\boldsymbol{x}_h) &= \|\boldsymbol{\omega}_l \cdot (\boldsymbol{y}_l - \boldsymbol{k}_{\boldsymbol{X}_l \boldsymbol{x}_h}(k_{\boldsymbol{x}_h \boldsymbol{x}_h} + \beta)^{-1} y_h)\|^2 \,, \\ &= \sum_{i=1}^{\mathrm{N}} \omega_{li}^2 (y_{li} - \frac{k(\boldsymbol{X}_{li}, \boldsymbol{x}_h)}{k_{\boldsymbol{x}_h \boldsymbol{x}_h} + \beta} y_h)^2 \,,\end{aligned} \tag{13}$$

where $k(\boldsymbol{X}_{li}, \boldsymbol{x}_h)$ measures the similarity between the i_{th} design of $\boldsymbol{X}_l$ and the high-scoring design $\boldsymbol{x}_h$. Recall that y_h represents the predefined target score and is generally much larger than y_{li}. By minimizing $\mathcal{L}_{h2l}(\boldsymbol{x}_h)$, we aim to find a high-scoring design $\boldsymbol{x}_h$ such that any design in the static dataset similar to the high scoring design is encouraged to have a high score prediction. In this way, $\boldsymbol{x}_h$ tries to incorporate as many high-scoring features from the static dataset as possible.

Optimization. Combining the two losses, the overall loss can be expressed as:

$$\begin{aligned}\mathcal{L}(\boldsymbol{X}_h) = \frac{1}{2}(&\|\boldsymbol{\omega}_h \cdot (\boldsymbol{y}_h - \boldsymbol{K}_{\boldsymbol{X}_h \boldsymbol{X}_l}(\boldsymbol{K}_{\boldsymbol{X}_l \boldsymbol{X}_l} + \beta \boldsymbol{I})^{-1} \boldsymbol{y}_l)\|^2 \\ &+ \|\boldsymbol{\omega}_l \cdot (\boldsymbol{y}_l - \boldsymbol{K}_{\boldsymbol{X}_l \boldsymbol{X}_h}(\boldsymbol{K}_{\boldsymbol{X}_h \boldsymbol{X}_h} + \beta \boldsymbol{I})^{-1} \boldsymbol{y}_h)\|^2) \,,\end{aligned} \tag{14}$$

where only the high-scoring designs $\boldsymbol{X}_h$ are learnable parameters in the whole procedure of BDI. We optimize $\boldsymbol{X}_h$ against $\mathcal{L}(\boldsymbol{X}_h)$ by gradient descent methods such as Adam [21].

4 Experiments

We conduct extensive experiments on design-bench [1], and aim to answer three research questions: (1) How does BDI compare with both recently proposed offline model-based algorithms and traditional algorithms? (2) Is every component necessary in BDI? (i.e., $\mathcal{L}_{h2l}(\boldsymbol{X}_h)$, $\mathcal{L}_{l2h}(\boldsymbol{X}_h)$, NTK.) (3) Is BDI robust to the hyperparameter choices including the predefined target score y_h, the weight parameter α and the number of steps T?

4.1 Dataset and Evaluation

To evaluate the effectiveness of BDI, we adopt the commonly used design-bench, including both continuous and discrete tasks, and the evaluation protocol as in the prior work [8].

Task overview. We conduct experiments on four continuous tasks: **(a)** Superconductor (SuperC) [2], where the aim is to design a superconductor with 86 continuous components to maximize critical temperature with 17010 designs; **(b)** Ant Morphology (Ant) [1, 22], where the goal is to design the morphology of a quadrupedal ant with 60 continuous components to crawl quickly with 10004 designs, **(c)** D'Kitty Morphology (D'Kitty) [1, 23], where the goal is to design the morphology of a quadrupedal D'Kitty with 56 continuous components to crawl quickly with 10004 designs; and **(d)** Hopper Controller (Hopper) [1], where the aim is to find a neural network policy parameterized by 5126 total weights to maximize return with 3200 designs. Besides the continuous tasks, we perform experiments on three discrete tasks: **(e)** GFP [3], where the objective is to find a length 238 protein sequence to maximize fluorescence with 5000 designs; **(f)** TF Bind 8 (TFB) [5], where the aim is to find a length 8 DNA sequence to maximize binding activity score with 32896 designs; and **(g)** UTR [6], where the goal is to find a length 50 DNA sequence to maximize expression level with $140,000$ designs. These tasks contain neither personally identifiable nor offensive information.

Evaluation. We follow the same evaluation protocol in [8]: we choose the top $\mathrm{N} = 128$ most promising designs for each method, and then report the 100th percentile normalized ground truth score as $y_n = \frac{y - y_{min}}{y_{max} - y_{min}}$ where y_{min} and y_{max} represent the lowest score and the highest score in the full unobserved dataset, respectively. The additional 50th percentile (median) normalized ground truth scores, used in the prior work [8], are also provided in Appendix A.1. To better measure the performance across multiple tasks, we further report the mean and median ranks of the compared algorithms over all 7 tasks.

4.2 Comparison Methods

We compare BDI with two groups of baselines: (i) sampling via a generative model; and (ii) gradient updating from existing designs. The generative model based methods learn a distribution of the high-scoring designs and then sample from the learned distribution. This group of methods includes: 1) CbAS [24]: trains a VAE model of the design distribution using designs with a score above a threshold and gradually adapts the distribution to the high-scoring part by increasing the threshold. 2) Auto.CbAS [25]: uses importance sampling to retrain a regression model using the design distribution introduced by CbAS; 3) MIN [20]: learns an inverse map from the score to the design and queries the inverse map by searching for the optimal score to obtain the optimal design. The latter group includes: 1) Grad: applies simple gradient ascent on existing designs to obtain new designs; 2) ROMA [7]: regularizes the smoothness of the DNN and then performs gradient ascent to obtain new designs; 3) COMs [8]: regularizes the DNN to assign lower scores to designs obtained during the gradient ascent process and leverages this model to obtain new designs by gradient ascent; 4) NEMO [9]: bounds the distance between the proxy and the objective function by normalized maximum likelihood and then performs gradient ascent. We discuss the NTK based Grad in Section 4.5.

Besides the recent work, we also compare BDI with several traditional methods described in [1]: 1) BO-qEI [26]: performs Bayesian Optimization to maximize the proxy, proposes designs via the quasi-Expected-Improvement acquisition function, and labels the designs via the proxy function; 2) CMA-ES [27]: an evolutionary algorithm gradually adapts the distribution via the covariance matrix towards the optimal design; 3) REINFORCE [28]: first learns a proxy function and then optimizes the distribution over the input space by leveraging the proxy and the policy-gradient estimator.

4.3 Training Details

We follow the training settings of [8] for all comparison methods if not specified. Recall that M represents the number of the high-scoring designs. We set $\mathrm{M} = 1$ in our experiments, which achieves expressive results, and discuss the M >1 scenario in Appendix $\mathrm{A}.2$. We set the regularization β as $1e^{-6}$ following [16]. We set the predefined target score y_h as a constant 10 across all tasks, which proves to be effective and robust in all cases. We set α as $1e^{-3}$ for all continuous tasks and as 0.0 for all discrete tasks, and set the number of iterations T to 200 in all experiments. We discuss the choices of y_h, α and T in Sec. 4.6 in detail. We adopt a 6-layer MLP (MultiLayer Perceptron) followed by ReLU for all gradient updating methods and the hidden size is set as 2048. We optimize $\mathcal{L}(\boldsymbol{X}_h)$ with the Adam optimizer [21] with a $1e^{-1}$ learning rate for discrete tasks and a $1e^{-3}$ learning rate for continuous tasks. We cite the results from [8] for the non-gradient-ascent methods including BO-qEI, CMA-ES[1], REINFORCE, CbAS, Auto.CbAS. For other methods, we run every setting over 8 trials and report the mean and the standard error. We use the NTK library [29] build on Jax [30] to conduct kernel-based experiments and use Pytorch [31] for other experiments. All Jax experiments are run on multiple CPUs within a cluster and all Pytorch experiments are run on one V100 GPU. We discuss the computational time details for all tasks in Appendix $\mathrm{A}.3$.

4.4 Results and Analysis

We report experimental results in Table 1 for continuous tasks and in Table 2 for discrete tasks where $\mathcal{D}$(**best**) represents the maximal score of the static dataset for each task. We bold the results within one standard deviation of the highest performance.

Results on continuous tasks. As shown in Table 1, BDI achieves the best results over all tasks. Compared with the naive Grad method, BDI achieves consistent gain over all four tasks, which suggests that BDI can align the high-scoring design with the static dataset and thus mitigate the out-of-distribution problem. As for COMs, ROMA and NEMO, which all impose a prior on the DNN model, they generally perform better than Grad but worse than BDI. This indicates that directly transforming the information of the static dataset into the high-scoring design is more effective than adding priors on the DNN in terms of mitigating the out-of-distribution problem. Another advantage of BDI is in its neural tangent kernel nature: 1) Real-world offline model-based optimization often involves small-scale datasets since the labeling cost of protein/DNA/robot is very high; 2) BDI is built on the neural tangent kernel, which generally performs better than finite neural networks on

[1]CMA-ES performs very differently on Ant Morphology and D'Kitty Morphology and the reason is that Ant Morphology is simpler and much more sensitive to initializations.

Table 1: Experimental results on continuous tasks for comparison.

Method	Superconductor	Ant Morphology	D'Kitty Morphology	Hopper Controller
$\mathcal{D}$(**best**)	0.399	0.565	0.884	1.0
BO-qEI	0.402 ± 0.034	0.819 ± 0.000	0.896 ± 0.000	0.550 ± 0.118
CMA-ES	0.465 ± 0.024	$\mathbf{1.214 \pm 0.732}$	0.724 ± 0.001	0.604 ± 0.215
REINFORCE	0.481 ± 0.013	0.266 ± 0.032	0.562 ± 0.196	-0.020 ± 0.067
CbAS	0.503 ± 0.069	0.876 ± 0.031	0.892 ± 0.008	0.141 ± 0.012
Auto.CbAS	0.421 ± 0.045	0.882 ± 0.045	0.906 ± 0.006	0.137 ± 0.005
MIN	0.452 ± 0.026	0.908 ± 0.020	$\mathbf{0.942 \pm 0.010}$	0.094 ± 0.113
Grad	0.483 ± 0.027	0.764 ± 0.047	0.888 ± 0.035	0.989 ± 0.362
COMs	0.487 ± 0.023	0.866 ± 0.050	0.835 ± 0.039	1.224 ± 0.555
ROMA	0.476 ± 0.024	0.814 ± 0.051	0.905 ± 0.018	1.849 ± 0.110
NEMO	0.488 ± 0.034	0.814 ± 0.043	0.924 ± 0.012	$\mathbf{1.959 \pm 0.159}$
BDI$_{\text{(ours)}}$	$\mathbf{0.520 \pm 0.005}$	$\mathbf{0.962 \pm 0.000}$	$\mathbf{0.941 \pm 0.000}$	$\mathbf{1.989 \pm 0.050}$

Table 2: Experimental results on discrete tasks & ranking on all tasks for comparison.

Method	GFP	TF Bind 8	UTR	Rank Mean	Rank Median
$\mathcal{D}$(**best**)	0.789	0.439	0.593		
BO-qEI	0.254 ± 0.352	0.798 ± 0.083	0.684 ± 0.000	8.6/11	9/11
CMA-ES	0.054 ± 0.002	0.953 ± 0.022	0.707 ± 0.014	5.7/11	6/11
REINFORCE	$\mathbf{0.865 \pm 0.000}$	0.948 ± 0.028	0.688 ± 0.010	7.4/11	9/11
CbAS	$\mathbf{0.865 \pm 0.000}$	0.927 ± 0.051	0.694 ± 0.010	4.6/11	5/11
Auto.CbAS	$\mathbf{0.865 \pm 0.000}$	0.910 ± 0.044	0.691 ± 0.012	6.0/11	7/11
MIN	$\mathbf{0.865 \pm 0.001}$	0.882 ± 0.020	0.694 ± 0.017	5.3/11	4/11
Grad	0.864 ± 0.001	0.491 ± 0.042	0.666 ± 0.013	7.9/11	8/11
COMs	0.861 ± 0.009	0.920 ± 0.043	0.699 ± 0.011	5.6/11	6/11
ROMA	0.558 ± 0.395	0.928 ± 0.038	0.690 ± 0.012	6.1/11	7/11
NEMO	0.150 ± 0.270	0.905 ± 0.048	0.694 ± 0.015	5.4/11	4/11
BDI$_{\text{(ours)}}$	0.864 ± 0.000	$\mathbf{0.973 \pm 0.000}$	$\mathbf{0.760 \pm 0.000}$	**1.9/11**	**1/11**

small-scale datasets [32]; 3) COMs, ROMA, and NEMO are built on a finite neural network and it is non-trivial to modify them to the neural tangent kernel version. The generative model based methods CbAS, Auto.CbAS and MIN perform poorly for high-dimensional tasks like Hopper (D=5126) since the high-dimensional data distributions are harder to model. Compared with generative model based methods, BDI not only achieves better results but is also much simpler.

Results on discrete tasks. As shown in Table 2, BDI achieves the best or equal-best performances in two of the three tasks, and is only marginally inferior in the third. This suggests that BDI is also a powerful approach in the discrete domain. For the GFP task, every design is a length 238 sequence of 20-categorical one-hot vectors and gradient updates that do not take into account the sequential nature may be less beneficial. This may explain why BDI does not perform as well on GFP.

Overall, BDI achieves the best result in terms of the ranking as shown in Table 2 and Figure 2, and attains the best performance on **6/7** tasks.

Table 3: Ablation studies on BDI components.

Task	D	BDI	w/o $\mathcal{L}_{l2h}$	w/o $\mathcal{L}_{h2l}$	NTK2NN	NTK2RBF
GFP	238	$\mathbf{0.864 \pm 0.000}$	0.860 ± 0.004	$\mathbf{0.864 \pm 0.000}$	0.252 ± 0.354	0.862 ± 0.003
TFB	8	0.973 ± 0.000	$\mathbf{0.984 \pm 0.000}$	0.973 ± 0.000	0.958 ± 0.025	0.925 ± 0.000
UTR	50	0.760 ± 0.000	0.636 ± 0.000	$\mathbf{0.777 \pm 0.000}$	0.699 ± 0.009	0.738 ± 0.000
SuperC	86	$\mathbf{0.520 \pm 0.005}$	0.511 ± 0.007	0.502 ± 0.007	0.450 ± 0.036	0.510 ± 0.012
Ant	60	$\mathbf{0.962 \pm 0.000}$	0.914 ± 0.000	0.933 ± 0.000	0.861 ± 0.019	0.927 ± 0.000
D'Kitty	56	$\mathbf{0.941 \pm 0.000}$	0.920 ± 0.000	$\mathbf{0.942 \pm 0.001}$	0.929 ± 0.010	0.938 ± 0.008
Hopper	5126	$\mathbf{1.989 \pm 0.050}$	1.935 ± 0.553	1.821 ± 0.223	0.503 ± 0.039	0.965 ± 0.103

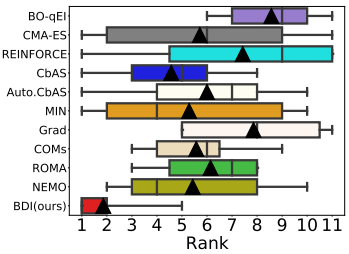

Figure 2: Rank minima and maxima are indicated by whiskers; vertical lines and black triangles represent medians and means.

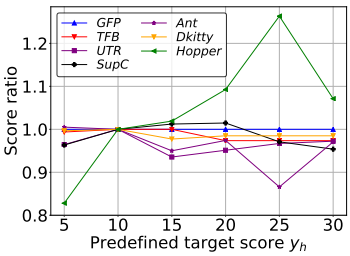

Figure 3: **The ratio of** the ground truth score of the design with y_h **to** the ground truth score with $y_h = 10$.

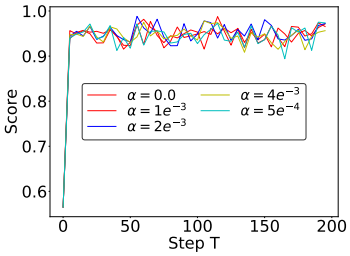

Figure 4: The ground truth score of the design as a function of the step T for different weight parameters α on Ant.

4.5 Ablation Studies

In this subsection, we conduct ablation studies on the components of BDI and aim to verify the effectiveness of $\mathcal{L}_{h2l}(\boldsymbol{X}_h)$, $\mathcal{L}_{l2h}(\boldsymbol{X}_h)$, and NTK. The baseline method is our proposed BDI and we remove each component to verify its effectiveness. We replace the NTK with its corresponding finite DNN (6-layer MLP followed by ReLU), which is trained on $(\boldsymbol{X}_l, \boldsymbol{y}_l)$ for Eq.(5) and $(\boldsymbol{X}_h, \boldsymbol{y}_h)$ for Eq.(7). Following [16], another alternative is to replace the NTK with RBF.

As shown in Table 3, BDI generally performs the best (at least second-best) across tasks, which indicates the importance of all three components.

Forward mapping. Removing $\mathcal{L}_{l2h}(\boldsymbol{X}_h)$ yields the best result 0.984 in the TFB task, but worsens the designs in other cases, which verifies the importance of the forward mapping. Incorporating the $\mathcal{L}_{l2h}(\boldsymbol{X}_h)$ loss term is similar to using the Grad method as we discuss in Section 3.1. We also run experiments on gradient ascent with NTK. This approach first builds a regression model using the NTK and then performs gradient ascent on existing designs to obtain new designs. Results are very similar to BDI without $\mathcal{L}_{h2l}(\boldsymbol{X}_h)$, so we do not report them here.

Backward mapping. Removing $\mathcal{L}_{h2l}(\boldsymbol{X}_h)$ decreases model performances in three tasks, and only improves it for the UTR task. This demonstrates the importance of the backward mapping. Although removing $\mathcal{L}_{h2l}$ leads to the best result 0.777 in UTR compared with 0.760 of BDI, this does not necessarily demonstrate that the removal of $\mathcal{L}_{h2l}$ is desirable. The 50th percentile of BDI is higher than that of BDI without $\mathcal{L}_{h2l}$ ($0.664 > 0.649$). Furthermore, we observe that the backward mapping is more effective for continuous tasks, possibly because we do not model the sequential nature of DNA and protein sequences. We follow the default procedure in [8] which maps the sequence design to real-valued logits of a categorical distribution for a fair comparison. Given a more effective discrete space modeling, the backward mapping might yield better results; we leave this to future work.

Neural tangent kernel. We replace the NTK with the corresponding finite DNN (NTK2NN) and the RBF kernel (NTK2RBF), and both replacements degrade the performance significantly, which verifies the effectiveness of the NTK. In addition, we can observe that NTK2NN performs relatively well on the low-dimensional task TFB (D=8) but poorly on high-dimensional tasks, especially for GFP ($\mathrm{D} = 238$) and Hopper ($\mathrm{D} = 5126$). It is likely that the high-dimensional designs $\boldsymbol{X}_h$ are more sensitive to the approximation in Eq. (9). Furthermore, BDI performs better than NTK2RBF and this can be explained as follows. Compared with the RBF kernel, the NTK enjoys the high expressiveness of a DNN [15] and represents a broad class of DNNs [16, 17], which can better extract the design features and enhance the generalization of the high-scoring designs.

4.6 Hyperparameter Sensitivity

In this subsection, we first study the sensitivity of BDI to the predefined target score y_h. Note that all algorithms in this subsection do not have access to the ground truth scores, which are only used for analysis. We expect this score to be larger than the maximal score of the static dataset since we want to obtain designs better than those in the static dataset. We first normalize all scores to have unit Gaussian statistics following [1, 8]. A small target score may not be able to identify a high-scoring design and a large score is too hard to reach. Thus we choose to set the target score y_h as 10 across all

Table 4: Comparison between data distillation and backward mapping.

Comparison	Data distillation	Backward mapping
Data $\boldsymbol{x}$	images in the training dataset $\mathcal{D}$	designs in the offline dataset $\mathcal{D}$
Label content	0-9 for the MNIST dataset	some measurement of protein/dna/robot
Label value	within the training dataset $\mathcal{D}$	larger than the max of the offline dataset $\mathcal{D}$
Loss function	cross-entropy for classification	mean squared error for regression
Task objective	distill $\mathcal{D}$ into a small subset	distill $\boldsymbol{x}$ to incorporate high-scoring features

tasks. In Figure 3, we report **the ratio of** the ground truth score of the design with y_h **to** the ground truth score of the design with $y_h = 10$ and the score details are in Appendix $\mathrm{A}.4$. All tasks are robust to the change of y_h. Although we set y_h as 10, which is smaller than the maximal score 10.2 for Hopper, BDI still yields a good result, which demonstrates the robustness of BDI. For Hopper, after we set y_h to a larger value, we can observe that the corresponding results become better.

Besides y_h, we also explore robustness to the choice of the weight parameter α and the number of steps T. We study the continuous task Ant and the discrete task TFB. As shown in Figure 4, we visualize the 100th percentile ground truth score of the Ant task as a function of T for $\alpha = 0.0, 5e^{-4}, 1e^{-3}, 2e^{-3}, 4e^{-3}$. We can observe that the behavior of BDI is robust to change of α within the range of evaluated values. BDI yields the high-scoring designs at early steps and then the solutions remain stable. This demonstrates the robustness of BDI to the choice of T. For the TFB task, we find the behavior of BDI is identical for different α and the behavior with respect to T is very similar to that of Ant. Thus we do not report details here. One possible reason for the insensitivity to α is that TFB is a discrete task and the discrete solution is more robust to α.

5 Related Work

Offline model-based optimization. Recent methods for offline model-based optimization can be divided into two categories: (i) obtaining designs from a generative model and (ii) applying gradient ascent on existing designs. In the first category, the methods in [24, 25] gradually adapt a VAE-based generative model towards the optimized design by leveraging a proxy function. Kumar et al. adopt an alternative approach in [20], learning a generator that maps the score to the design. The generative model based methods often require careful tuning to model the high-scoring designs [8].

Gradient-based methods have received substantial attention recently because they can leverage DNN models to produce improved designs. A challenge in the naive application of such methods is out-of-distribution designs, for which the trained DNN model produces inaccurate score predictions. Several approaches have been proposed to address this. Trabucco et al. add a regularizing term for conservative objective modeling [8, 33]. In [9], Fu et al. leverage normalized maximum likelihood to bound the distance between the proxy function and the ground truth. Yu et al. overcome the brittleness of the proxy function by utilizing a local smoothness prior [7]. Our work falls into the gradient-based line of research, but rather than imposing a prior on the *model*, we propose the bidirectional mapping to ensure that the proposed designs can be used to predict the scores of the *data* and vice versa, thus ensuring that the high-scoring designs are more aligned with the static dataset. Similar to our backward mapping constraining designs, BCQ [34] constrains the state-action pairs to be contained in the support of the static dataset. Our proposed backward mapping is different in its implementation and also enables the model to generate designs outside the training distribution.

Data distillation. Data distillation [18] aims to distill a large training set into a small one, which enables efficient retraining. Building upon [18], the methods in [35, 36] propose to distill labels instead of images. Approaches in [16, 37] introduce a meta-learning algorithm powered by NTK, which significantly improves the performance. The backward mapping of BDI is inspired by [16, 37] and the difference is that BDI is handling a regression problem with a predefined target score larger than the maximal score of the static dataset. In Table 4 we provided a detailed comparison of backward mapping and data distillation; both adopt a bi-level framework [38]. This bi-level framework can also be used to learn other hyperaparameters [17, 39–43].

6 Conclusion and discussion

In this paper, we address offline model-based optimization and propose a novel approach that involves bidirectional score predictions. BDI requires the proposed high-scoring designs and the static offline dataset to predict each other, which effectively ameliorates the out-of-distribution problem. The finite-width DNN can only yield an approximate loss function, which leads to a significant deterioration of the design quality, especially for high-dimensional tasks. We thus adopt an infinite-width DNN and employ the NTK to yield a closed-form solution, leading to better designs. Experimental results verify the effectiveness of BDI, demonstrating that it outperforms state-of-the-art approaches.

Limitations. Although BDI has been demonstrated to be effective in tasks from a wide range of domains, some evaluations are not performed with a realistic setup. For example, in some of our experiments, such as the GFP [3] task, we follow the prior work and use a transformer model which is pre-trained on a large dataset as the evaluation oracle. However, this may not reflect real-world behavior and the evaluation protocol can be further improved by close collaboration with domain experts [44]. Overall, we still believe BDI can generalize well to these cases since BDI has a simple formulation and proves to be effective and robust for the wide range of tasks in the design-bench.

Negative impacts. Designing a new object or entity with desired properties is a double-edged sword. On one hand, it can be beneficial to society, with an example being drug discovery to cure unsolved diseases. On the other hand, if these techniques are acquired by dangerous people, they can also be used to design biochemicals to harm our society. Researchers should pay vigilant attention to ensure their research does end up being used positively for the social good.

7 Acknowledgement

We thank Aristide Baratin from Mila and Greg Yang from Microsoft Research for their helpful discussions on the neural tangent kernel and thank Zixuan Liu from the University of Washington for his helpful suggestions on the paper writing. This research was supported in part by funding from the Fonds de recherche du Québec – Nature et technologies.

References

[1] Brandon Trabucco, Xinyang Geng, Aviral Kumar, and Sergey Levine. Design-bench: Benchmarks for data-driven offline model-based optimization. arXiv preprint arXiv:2202.08450, 2022.

[2] Kam Hamidieh. A data-driven statistical model for predicting the critical temperature of a superconductor. Computational Materials Science, 2018.

[3] Karen S Sarkisyan et al. Local fitness landscape of the green fluorescent protein. Nature, 2016.

[4] Christof Angermueller, David Dohan, David Belanger, Ramya Deshpande, Kevin Murphy, and Lucy Colwell. Model-based reinforcement learning for biological sequence design. In Proc. Int. Conf. Learning Rep. (ICLR), 2019.

[5] Luis A Barrera et al. Survey of variation in human transcription factors reveals prevalent dna binding changes. Science, 2016.

[6] Paul J Sample, Ban Wang, David W Reid, Vlad Presnyak, Iain J McFadyen, David R Morris, and Georg Seelig. Human 5 UTR design and variant effect prediction from a massively parallel translation assay. Nature Biotechnology, 2019.

[7] Sihyun Yu, Sungsoo Ahn, Le Song, and Jinwoo Shin. Roma: Robust model adaptation for offline model-based optimization. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2021.

[8] Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Conservative objective models for effective offline model-based optimization. In Proc. Int. Conf. Learning Rep. (ICLR), 2021.

[9] Justin Fu and Sergey Levine. Offline model-based optimization via normalized maximum likelihood estimation. Proc. Int. Conf. Learning Rep. (ICLR), 2021.

[10] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2021.

[11] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2018.

[12] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2019.

[13] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as Gaussian processes. Proc. Int. Conf. Learning Rep. (ICLR), 2017.

[14] Greg Yang and Etai Littwin. Tensor programs iib: Architectural universality of neural tangent kernel training dynamics. In Proc. Int. Conf. Machine Lea. (ICML), 2021.

[15] Jaehoon Lee, Samuel Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2020.

[16] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. Proc. Int. Conf. Learning Rep. (ICLR), 2020.

[17] Chia-Hung Yuan and Shan-Hung Wu. Neural tangent generalization attacks. In Proc. Int. Conf. Machine Lea. (ICML), 2021.

[18] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. arXiv preprint arXiv:1811.10959, 2018.

[19] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In Proc. Int. Conf. Machine Lea. (ICML), 2018.

[20] Aviral Kumar and Sergey Levine. Model inversion networks for model-based optimization. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2020.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Proc. Int. Conf. Learning Rep. (ICLR), 2015.

[22] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

[23] Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. Robel: Robotics benchmarks for learning with low-cost robots. In Conf. on Robot Lea. (CoRL), 2020.

[24] David Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. In Proc. Int. Conf. Machine Lea. (ICML), 2019.

[25] Clara Fannjiang and Jennifer Listgarten. Autofocused oracles for model-based design. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2020.

[26] James T Wilson, Riccardo Moriconi, Frank Hutter, and Marc Peter Deisenroth. The reparameterization trick for acquisition functions. arXiv preprint arXiv:1712.00424, 2017.

[27] Nikolaus Hansen. The CMA evolution strategy: A comparing review. In Jose A. Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea, editors, Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms, pages 75–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[28] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 1992.

[29] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In Proc. Int. Conf. Learning Rep. (ICLR), 2020.

[30] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. Jax: composable transformations of python+ numpy programs. URL http://github.com/google/jax, 2020.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2019.

[32] Sanjeev Arora, Simon S. Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In Proc. Int. Conf. Machine Lea. (ICML), 2020.

[33] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2019.

[34] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In Proc. Int. Conf. Machine Lea. (ICML), 2019.

[35] Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Flexible dataset distillation: Learn labels instead of images. arXiv preprint arXiv:2006.08572, 2020.

[36] Ilia Sucholutsky and Matthias Schonlau. Soft-label dataset distillation and text dataset distillation. In Proc. Int. Joint Conf. on Neur. Net. (IJCNN), 2021.

[37] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2021.

[38] Can Chen, Xi Chen, Chen Ma, Zixuan Liu, and Xue Liu. Gradient-based bi-level optimization for deep learning: A survey. arXiv preprint arXiv:2207.11719, 2022.

[39] Zhiting Hu, Bowen Tan, Russ R Salakhutdinov, Tom M Mitchell, and Eric P Xing. Learning data manipulation for augmentation and weighting. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2019.

[40] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In Proc. Int. Conf. Machine Lea. (ICML), 2018.

[41] Can Chen, Chen Ma, Xi Chen, Sirui Song, Hao Liu, and Xue Liu. Unbiased implicit feedback via bi-level optimization. arXiv preprint arXiv:2206.00147, 2022.

[42] Can Chen, Shuhao Zheng, Xi Chen, Erqun Dong, Xue Steve Liu, Hao Liu, and Dejing Dou. Generalized data weighting via class-level gradient manipulation. Proc. Adv. Neur. Inf. Proc. Syst (NeurIPS), 2021.

[43] Can Chen, Jingbo Zhou, Fan Wang, Xue Liu, and Dejing Dou. Structure-aware protein self-supervised learning. arXiv preprint arXiv:2204.04213, 2022.

[44] Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Artificial intelligence foundation for therapeutic science. Nature Chemical Biology, 2022.

[45] Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V Davuluri. Dnabert: pre-trained bidirectional encoder representations from transformers model for dna-language in genome. Bioinformatics, 2021.

[46] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rihawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, et al. ProtTrans: towards cracking the language of life's code through self-supervised deep learning and high performance computing. arXiv preprint arXiv:2007.06225, 2020.

[47] Courtney E Gonzalez and Marc Ostermeier. Pervasive pairwise intragenic epistasis among sequential mutations in tem-1 β-lactamase. Journal of molecular biology, 2019.

[48] Elad Firnberg, Jason W Labonte, Jeffrey J Gray, and Marc Ostermeier. A comprehensive, high-resolution map of a gene's fitness landscape. Molecular biology and evolution, 2014.

[49] Dinghuai Zhang, Jie Fu, Yoshua Bengio, and Aaron Courville. Unifying likelihood-free inference with black-box optimization and beyond. In Proc. Int. Conf. Learning Rep. (ICLR), 2021.

[50] Emily E Wrenbeck, Laura R Azouz, and Timothy A Whitehead. Single-mutation fitness landscapes for an enzyme on multiple substrates reveal specificity is globally encoded. Nature Communications, 2017.

[51] Justin R Klesmith, John-Paul Bacik, Ryszard Michalczyk, and Timothy A Whitehead. Comprehensive sequence-flux mapping of a levoglucosan utilization pathway in e. coli. ACS Synthetic Biology, 2015.

[52] Jochen Weile, Song Sun, Atina G Cote, Jennifer Knapp, Marta Verby, Joseph C Mellor, Yingzhou Wu, Carles Pons, Cassandra Wong, Natascha van Lieshout, et al. A framework for exhaustively mapping functional missense variants. Molecular Systems Biology, 2017.

[53] Zuobai Zhang, Minghao Xu, Arian Jamasb, Vijil Chenthamarakshan, Aurelie Lozano, Payel Das, and Jian Tang. Protein representation learning by geometric structure pretraining. arXiv preprint arXiv:2203.06125, 2022.

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 in Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We discuss negative impacts in Section 6.
   (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? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
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] They are in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All of these can in be found in Section 4.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We run every setting 8 times and report the mean and standard deviation in the tables.
   (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] It is in Section 4.3.
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 cite them [1].
   (b) Did you mention the license of the assets? [Yes] We mention the license in the README.md of our code in the supplemental material.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include our code in the supplemental material as a new asset and we will release this new asset upon paper acceptance.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] The assets are publicly available and free for research use.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] We use design-bench, which does not contain personally identifiable information nor offensive content, as illustrated in Section 4.1.

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 Appendix

A.1 Additional Results on 50th Percentile Scores

In the main paper, we have provided results for the 100th percentile metric and here we provide additional results on the 50th percentile metric, also used in the prior work design-bench, to verify the effectiveness of BDI. Table 5 and Table 6, report 50th percentile results for continuous tasks and discrete tasks, respectively. We can observe from Table 6 that BDI achieves the best result in terms of ranking. Note that there is no randomness for BDI by the nature of the method, once the initial design has been selected. To quantify the robustness of BDI to this starting design, we perturb each initial design by performing a small gradient ascent step according to a trained DNN model on the design. The mean and standard deviation are obtained over 8 different seeds. We can see that BDI is robust to the randomness.

A.2 Analysis of multiple high-scoring designs

We now discuss the high-scoring design number M >1 scenarios and consider two cases: (a) all M high-scoring designs are learnable; (b) only one high-scoring design is learnable and the other $\mathrm{M}-1$ designs are fixed. In both cases, we set the predefined target score y_h as 10. For case (a), we first choose a design from the static dataset and perform $\mathrm{M}-1$ gradient ascent steps on the design to obtain the $\mathrm{M}-1$ designs. Then BDI is performed between the static dataset and the M designs. Finally, the design with the maximal predicted score among the M designs is chosen for evaluation. For case (b), we choose $\mathrm{M}-1$ designs from the static dataset along with their ground-truth scores and then perform our BDI algorithm.

Table 5: Experimental results of 50th on continuous tasks for comparison.

Method	Superconductor	Ant Morphology	D'Kitty Morphology	Hopper Controller
$\mathcal{D}$(**best**)	0.399	0.565	0.884	1.000
BO-qEI	0.300 ± 0.015	0.567 ± 0.000	$\mathbf{0.883 \pm 0.000}$	0.343 ± 0.010
CMA-ES	0.379 ± 0.003	-0.045 ± 0.004	0.684 ± 0.016	-0.033 ± 0.005
REINFORCE	$\mathbf{0.463 \pm 0.016}$	0.138 ± 0.032	0.356 ± 0.131	-0.064 ± 0.003
CbAS	0.111 ± 0.017	0.384 ± 0.016	0.753 ± 0.008	0.015 ± 0.002
Auto.CbAS	0.131 ± 0.010	0.364 ± 0.014	0.736 ± 0.025	0.019 ± 0.008
MIN	0.325 ± 0.021	$\mathbf{0.606 \pm 0.025}$	$\mathbf{0.886 \pm 0.003}$	0.027 ± 0.115
Grad	0.400 ± 0.059	0.311 ± 0.046	0.736 ± 0.168	0.195 ± 0.086
COMs	0.365 ± 0.045	0.386 ± 0.083	0.685 ± 0.022	0.322 ± 0.032
ROMA	0.354 ± 0.014	0.383 ± 0.029	0.775 ± 0.009	0.358 ± 0.011
NEMO	0.359 ± 0.021	0.390 ± 0.028	0.800 ± 0.020	0.345 ± 0.034
BDI$_{(\text{ours})}$	0.408 ± 0.012	0.569 ± 0.000	0.875 ± 0.001	$\mathbf{0.419 \pm 0.003}$

We report the results of cases (a) and (b) in Table 7 and Table 8, respectively. In both cases the M >1 settings work well but are generally worse than $\mathrm{M} = 1$. One possible explanation is that BDI cannot distill sufficient information from the static dataset into the high-scoring design when the other $\mathrm{M} - 1$ designs also contribute to the backward mapping. As a result, the $\mathrm{M} = 1$ setting yields a better result.

A.3 Computational Complexity

The only learnable part of BDI is $\boldsymbol{X}_h$. Other coefficients, like $(\boldsymbol{K}_{\boldsymbol{X}_l\boldsymbol{X}_l} + \beta\boldsymbol{I})^{-1}\boldsymbol{y}_l$ in Eq. (12), can be pre-computed. Thus the computational time of BDI consists of two parts: pre-computation and gradient update. The main bottleneck of pre-computation lies in $(\boldsymbol{K}_{\boldsymbol{X}_l\boldsymbol{X}_l} + \beta\boldsymbol{I})^{-1}\boldsymbol{y}_l$, which results in the $\mathcal{O}(\mathrm{N}^3)$ time complexity, where N is the size of the static dataset $\mathcal{D}$. Note that the precomputation time includes the kernel matrix construction. The N of the static dataset usually is not large which means the computational time is usually reasonable, especially considering the potentially enormous cost of evaluating the unknown objective function later in the design cycle.

In many real-world settings, for example, those that involve biological or chemical experiments, the majority of the time in the production cycle will be spent on evaluating the unknown objective function. As a result, the time difference between the methods for obtaining the high score design is less significant in a real production environment.

As for the gradient update, its computation bottleneck lies in the forward and backward passes of NTK computation. We report the details of BDI computational time on different tasks in Table 9. All experiments are performed on CPUs in the same cluster. For other gradient-based methods, the computational time is on the same scale as BDI.

A.4 Predefined Target Scores

We report the unnormalized experimental results on different predefined target scores in Table 10 and these results correspond to Figure 3 in the main paper. Recall that the ground truth score returned by the evaluation oracle is y and we compute the normalized ground truth score as $y_n = \frac{y - y_{min}}{y_{max} - y_{min}}$ where y_{min} and y_{max} represent the lowest score and the highest score in the full unobserved dataset, respectively. We also report the normalized experimental results in Table 11. We can observe all tasks are robust to y_h. Although we set y_h as 10, which is smaller than the maximal score 10.2 for Hopper, BDI still yields a good result, which demonstrates the robustness of BDI.

A.5 Considering Sequential Nature

We adopt DNABert [45] and ProtBert [46] to model the sequential nature of DNA/protein sequences. We conduct an ablation study, removing the backward mapping to verify its effectiveness. We also remove the forward mapping to assess its importance. The following experiments demonstrate our intuition: **backward mapping matters**.

For the DNA experiments, we consider the task with an exact oracle: TFBind8, and obtain the 100th percentile results in Table 12. We can see that both the backward mapping and the forward mapping

Table 6: Experimental results of 50th on discrete tasks & ranking on all tasks for comparison.

Method	GFP	TF Bind 8	UTR	Rank Mean	Rank Median
$\mathcal{D}$(**best**)	0.789	0.439	0.593		
BO-qEI	0.246 ± 0.341	0.439 ± 0.000	0.571 ± 0.000	$6.0/11$	$5/11$
CMA-ES	0.047 ± 0.000	0.537 ± 0.014	0.612 ± 0.014	$7.1/11$	$10/11$
REINFORCE	0.844 ± 0.003	0.462 ± 0.021	0.568 ± 0.017	$7.4/11$	$10/11$
CbAS	$\mathbf{0.852 \pm 0.004}$	0.428 ± 0.010	0.572 ± 0.023	$7.4/11$	$9/11$
Auto.CbAS	0.848 ± 0.007	0.419 ± 0.007	0.576 ± 0.011	$7.9/11$	$8/11$
MIN	0.427 ± 0.012	0.421 ± 0.015	0.586 ± 0.007	$6.0/11$	$7/11$
Grad	0.836 ± 0.000	0.439 ± 0.000	0.611 ± 0.000	$5.4/11$	$5/11$
COMs	0.737 ± 0.262	0.512 ± 0.051	0.608 ± 0.012	$5.3/11$	$5/11$
ROMA	0.541 ± 0.382	0.439 ± 0.000	0.592 ± 0.005	$5.4/11$	$5/11$
NEMO	0.146 ± 0.260	0.439 ± 0.000	0.590 ± 0.007	$5.4/11$	$5/11$
BDI$_{(\mathrm{ours})}$	$\mathbf{0.860 \pm 0.010}$	$\mathbf{0.595 \pm 0.000}$	$\mathbf{0.664 \pm 0.000}$	**1.6/11**	**1/11**

Table 7: Experimental results on different Ms with all high-scoring designs learnable.

Task	$\mathcal{D}_{\textbf{best}}$	M = 1	M = 2	M = 3	M = 4
GFP	0.789	**0.864**	**0.864**	**0.864**	**0.864**
TFB	0.439	**0.973**	**0.973**	**0.973**	**0.973**
UTR	0.593	0.760	**0.762**	0.757	0.757
SuperC	0.399	**0.520**	0.502	0.485	0.510
Ant	0.565	**0.962**	0.323	0.275	0.277
D'Kitty	0.884	**0.941**	0.717	0.718	0.718
Hopper	1.000	**1.989**	1.412	1.299	1.319

Table 8: Experimental results on different Ms with one high-scoring design learnable.

Task	$\mathcal{D}_{\textbf{best}}$	M = 1	M = 2	M = 3	M = 4
GFP	0.789	**0.864**	0.854	0.857	0.858
TFB	0.439	**0.973**	0.779	0.934	0.934
UTR	0.593	**0.760**	0.728	0.720	0.720
SuperC	0.399	**0.520**	0.508	0.483	0.485
Ant	0.565	**0.962**	0.169	0.174	0.214
D'Kitty	0.884	**0.941**	0.721	0.720	0.719
Hopper	1.000	**1.989**	1.378	1.815	1.354

Table 9: Computational time of BDI on different tasks.

Task	GFP	TFB	UTR	SuperC	Ant	D'Kitty	Hopper
Size N	5000	32896	140000	17010	10004	10004	3200
Pre(mins)	0.2	4.1	64.5	1.2	0.5	0.5	0.1
Grad(mins)	67.1	9.3	93.6	9.1	7.6	9.8	54.6
Total(mins)	67.3	13.4	158.1	10.3	8.1	10.3	54.7

Table 10: Experimental results on different predefined target scores.

Task	$\mathcal{D}_{\textbf{best}}$	5	10	15	20	25	30
GFP	1.444	**6.294**	**6.294**	**6.294**	**6.294**	**6.294**	**6.294**
TFB	1.480	8.409	**8.462**	**8.462**	8.239	8.239	8.239
UTR	7.116	8.796	**9.120**	8.532	8.676	8.820	8.868
SuperC	2.652	3.609	3.746	3.792	**3.801**	3.637	3.573
Ant	1.756	**5.554**	5.526	5.250	5.383	4.785	5.364
D'Kitty	0.934	1.190	**1.194**	1.167	1.176	1.176	1.176
Hopper	10.221	19.797	23.906	24.363	26.121	**30.203**	25.609

Table 11: Experimental results on different predefined target scores (normalized).

Task	$\mathcal{D}_{\textbf{best}}$	5	10	15	20	25	30
GFP	0.789	**0.864**	**0.864**	**0.864**	**0.864**	**0.864**	**0.864**
TFB	0.439	0.969	**0.973**	**0.973**	0.956	0.956	0.956
UTR	0.593	0.733	**0.760**	0.711	0.723	0.735	0.739
SuperC	0.399	0.505	0.524	0.525	**0.526**	0.508	0.501
Ant	0.565	**0.965**	0.962	0.933	0.947	0.884	0.945
D'Kitty	0.884	0.940	**0.941**	0.935	0.937	0.937	0.937
Hopper	1.000	1.692	1.989	2.022	2.149	**2.444**	2.112

Table 12: Ablation studies on BDI components considering sequential nature.

Task	TFB	TFB(reduce)	LACT	AMP	ALIP	LEV	SUMO
BDI	**0.986**	**0.957**	**0.820**	**0.772**	**0.813**	**0.859**	**1.489**
w/o $\mathcal{L}_{l2h}$	0.954	0.849	-0.134	0.765	0.659	0.577	0.773
w/o $\mathcal{L}_{h2l}$	0.952	0.900	0.550	0.753	0.699	0.556	0.596

Table 13: Comparison between BDI and pure maximization.

Task	GFP	TFB	UTR	SuperC	Ant	D'Kitty	Hopper
Best Result	**0.865**	**0.973**	**0.760**	**0.520**	**0.962**	**0.941**	**1.989**
Our BDI	0.864	0.973	0.760	0.520	0.962	0.941	1.989
Pure Maxi	0.520	0.911	0.937	1.830	0.864	0.956	0.734

matter in BDI. We further consider a harder task setting TFBind8(reduce) or TFBind8(r): reduce the size of the offline dataset from 32896 to 5000. The results verify the effectiveness and the robustness of bidirectional mappings.

For the protein experiments, we first consider the GFP task we used in our paper, and obtain the same 0.864 results for all variants, which is also mentioned in [1] "lack in-distinguishable results across all methods". To further investigate this problem, we introduce 5 new protein tasks used in recent works: LACT, AMP, ALIP, LEV, and SUMO, and present the task details and the results in the following.

- LACT: We measure the thermodynamic stability of the TEM-1 β-Lactamase protein sequence. The oracle is trained on 17857 samples from [47, 48]. The bottom half of these samples are used for offline algorithms.
- AMP: Antimicrobial peptides are short protein sequences that act against pathogens. We use the 6760 AMPs from [49] to train the oracle and the bottom half of the AMPs for the offline algorithms.
- ALIP: We measure the enzyme activity of the Aliphatic Amide Hydrolase sequence following [50]. We use the 6629 samples from [50] to train the oracle and the bottom half of them for the offline algorithms.
- LEV: In an ATP-dependent reaction, LG can be converted by Levoglucosan kinase to the glucose-6-phosphate. Following [51], we measure the enzyme activity of this reaction. All 7891 protein sequences are used to train the oracle and the bottom half are for the offline algorithms.
- SUMO: Following [52], we measure the growth rescue rate of human SUMO E2 conjugase protein. Around 2000 samples are used to train the oracle and the bottom half are used for the offline algorithms.

As shown in Table 12, these results demonstrate the effectiveness of our backward mapping and forward mapping. Besides the sequential nature, the structural information [53, 43] can also be considered for protein design and we leave it to future work.

A.6 Comparison between Maximization and Regression

We change the regression to the pure maximization (Pure Maxi) of the predictions of the learned objective function and report the experimental results on all tasks.

As we can see in Table 13, Pure Maxi results are generally worse than the regression form and we conjecture that the advantage of the regression form arises due to the consistency between the forward mapping and the backward mapping which both use the same predefined target score y_h.

A.7 Varying Weights

We simply assign the forward mapping and the backward mapping equal weight since the two terms are symmetric and thus viewed as equally important. In this subsection, we view the weight term of the backward mapping as a hyperparameter and study the hyperparameter sensitivity on Ant and TFB as we did in Sec 4.6.

Table 14: Experimental results on different predefined target scores (normalized).

Task/Weight	0.00	0.50	0.80	0.90	**1.00**	1.10	1.20	1.50	2.00
Ant	0.933	0.950	0.970	0.944	0.962	0.943	0.956	0.941	0.938
TFB	0.973	0.973	0.973	0.973	0.973	0.973	0.973	0.973	0.973

Table 15: Analysis of backward mapping.

Task	BDI	forward mapping	backward mapping
Ant	0.0752	0.0739	0.0808
TFB	0.2646	0.2395	0.3570

For the ANT task, we find the results generally improve for any weight greater than zero. This demonstrates the effectiveness of backward mapping.

For the TFB task, we find the behavior of BDI is identical for different weight terms. One possible reason is that BDI here does not consider the sequential nature of DNA and thus the backward mapping cannot provide significant performance gain as we have discussed before.

A.8 Analysis of Backward Mapping

In this subsection, we provide more visualization and analysis to verify the effectiveness of the backward mapping. We use the NTK $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to measure the similarity between two points and compute the similarity between the generated high-scoring design $\boldsymbol{x}_h$ and the high-scoring designs $\boldsymbol{X}$ in the offline dataset. This is defined as:

$$simi(\boldsymbol{x_h}, \boldsymbol{X}) = mean(k(\boldsymbol{x}_h, \boldsymbol{X})). \tag{15}$$

As shown in Table 15, we find that $simi(\boldsymbol{x}_h^{backward}, \boldsymbol{X})$ >$simi(\boldsymbol{x}_h^{bdi}, \boldsymbol{X})$ >$simi(\boldsymbol{x}_h^{forward}, \boldsymbol{X})$. This can be explained as follows. Since the backward mapping extracts the high-scoring features from the offline dataset, $\boldsymbol{x}_h^{backward}$ is the closest to the high-scoring designs X in the offline dataset and thus $simi(\boldsymbol{x}_h^{backward}, \boldsymbol{X})$ is large. While encouraging the design $\boldsymbol{x}_h^{forward}$ to explore a high score, the forward mapping alone leads to the design far from the offline dataset, and thus $simi(\boldsymbol{x}_h^{forward}, \boldsymbol{X})$ is the smallest. BDI can explore a high score (forward mapping) and stay close to the offline dataset (backward mapping), which leads to $simi(\boldsymbol{x}_h^{bdi}, \boldsymbol{X})$ between $simi(\boldsymbol{x}_h^{forward}, \boldsymbol{X})$ and $simi(\boldsymbol{x}_h^{backward}, \boldsymbol{X})$.