

# Offline Model-Based Optimization via Policy-Guided Gradient Search

Yassine Chemingui, Aryan Deshwal, Trong Nghia Hoang, Janardhan Rao Doppa

School of EECS, Washington State University  
{yassine.chemingui, aryan.deshwal, trongnghia.hoang, jana.doppa}@wsu.edu

## Abstract

Offline optimization is an emerging problem in many experimental engineering domains including protein, drug or aircraft design, where online experimentation to collect evaluation data is too expensive or dangerous. To avoid that, one has to optimize an unknown function given only its offline evaluation at a fixed set of inputs. A naive solution to this problem is to learn a surrogate model of the unknown function and optimize this surrogate instead. However, such a naive optimizer is prone to erroneous overestimation of the surrogate (possibly due to over-fitting on a biased sample of function evaluation) on inputs outside the offline dataset. Prior approaches addressing this challenge have primarily focused on learning robust surrogate models. However, their search strategies are derived from the surrogate model rather than the actual offline data. To fill this important gap, we introduce a new *learning-to-search* perspective for offline optimization by reformulating it as an offline reinforcement learning problem. Our proposed policy-guided gradient search approach explicitly learns the best policy for a given surrogate model created from the offline data. Our empirical results on multiple benchmarks demonstrate that the learned optimization policy can be combined with existing offline surrogates to significantly improve the optimization performance.

## Introduction

Many science and engineering applications involve optimizing expensive-to-evaluate black-box functions over complex design spaces (Wang et al. 2023). Some prototypical examples include design optimization over input space of candidate proteins (Gao et al. 2020), molecules (Deshwal, Simon, and Doppa 2021), drugs (Schneider et al. 2020), hardware architectures (Huang et al. 2021; Deshwal et al. 2019), and superconducting materials (Fannjiang and Listgarten 2020). In such applications, evaluating a candidate input involves performing a lab experiment or expensive computational simulation. These problems are referred to as expensive black-box optimization (BBO). One standard framework to solve expensive BBO problems is Bayesian optimization where we iteratively query the black-box function’s evaluation for inputs recommended by a surrogate model, whose accuracy is continuously improved via learning from such

input-output pairs (Shahriari et al. 2015; Eriksson et al. 2019; Deshwal and Doppa 2021).

However, in some real-world scenarios where the overhead and cost of setting up experiments are prohibitively expensive (e.g., wet lab experiments that often require expensive materials and equipment), it becomes impractical to consider black-box optimization in the online setting. Instead, a more practical setting is to assume access to an existing database of previously collected input-output pairs, and consider solving this problem in an offline manner. This problem setting is termed *offline model-based BBO* and was first introduced in the recent work of (Kumar and Levine 2020; Trabucco et al. 2021). The goal of optimization method is to leverage this offline training data to uncover optimal designs from the given input space. There are two main interrelated challenges for effectively solving the offline optimization problem. First, how to learn surrogate models from the offline dataset which are robust on inputs outside the offline dataset. Second, how to create effective search strategies beyond the local neighborhood of training data to find high-quality inputs. These challenges are often amplified due to the complexity of optimization problems including high-dimensional search spaces, highly sensitive objective functions where similar inputs might not induce similar outputs, and sparsity of the available data.

Prior work on offline optimization (see Section ) can be grouped into two broad categories based on the algorithmic design choices to address these two challenges. The first family learns a generative model of the input distribution along side the surrogate model to characterize a trust region from which high-performing inputs can be sampled directly (Fannjiang and Listgarten 2020; Brookes, Park, and Listgarten 2019). However, generative models cannot be robustly learned on high-dimensional search spaces with sparse training data and domain knowledge required to create valid trust regions may not be available. The second family learns a surrogate model from the offline data which is then optimized directly using gradient updates (Trabucco et al. 2021; Yu et al. 2021a; Fu and Levine 2021). Conservative regularizers are typically designed to avoid overestimation for inputs which are far away from the offline training data. However, surrogate models, such as deep neural networks, can be non-smooth which could result in highly sub-optimal solutions for a fixed gradient search strategy. This

motivates the following question: *How can we learn policies to effectively guide the gradient-based search process in input regions outside the offline data?*

This paper answers the above question by introducing a new learning-to-search perspective and a principled approach referred to as *Policy-guided Gradient Search (PGS)* for solving offline optimization problems. PGS is inspired by the prior successes of integrating learning into search procedures (Daumé, Langford, and Marcu 2009; Doppa, Fern, and Tadepalli 2014) and learned optimizers (Flennerhag et al. 2019). The key idea behind PGS is to reformulate the step-size in the standard gradient update into a direction vector. Such direction vector can be viewed as an output  $\alpha = \pi(x_k)$  of a guiding policy  $\pi$  to direct the search space exploration from  $x_k$  in the direction of the high-performing input regions. PGS can be seen as a mechanism to correct the surrogate model’s gradients with respect to Oracle search. We formulate the policy learning task as an instance of offline reinforcement learning (RL) problem and provide an effective strategy for synthesizing trajectories from the offline dataset which is critical for creating high-performing policies. This reduction approach allows us to leverage the large body of existing work on offline RL to effectively solve offline optimization problems. Hyper-parameter selection is a challenging task for offline optimization and none of the prior methods provide any procedure. To address this challenge, we propose an approach referred to as offline state estimation via latent embeddings using only the offline dataset. Empirical evaluation on diverse optimization tasks from the design-bench benchmark (Trabucco et al. 2022) demonstrates that PGS performs better than prior methods on many tasks and ablation analysis shows the benefits of our algorithm design choices.

**Contributions:** The key contribution of this paper is the development and evaluation of the PGS approach for solving offline optimization problems. Specific contributions are:

- Learning-to-search formulation to guide gradient search and reducing policy learning to offline RL.
- Trajectory synthesis for offline RL based policy learning and hyperparameter tuning using trajectory embeddings.
- Empirical evaluation and ablation analysis of PGS on the design-bench benchmark. The code for PGS is publicly available at <https://github.com/yassineCh/PGS>.

## Problem Setup

Suppose  $\mathcal{X}$  is an input space where each  $x \in \mathcal{X}$  is a  $d$ -dimensional candidate input. Let  $f : \mathcal{X} \mapsto \mathbb{R}$  be an unknown, expensive real-valued objective function which can evaluate any given input  $x \in \mathcal{X}$  to produce output  $y = f(x)$ . For example, in drug design application,  $f(x)$  corresponds to running a physical lab experiment. Similarly, in hardware optimization application,  $f(x)$  call involves performing an expensive simulation to mimic the real hardware. Our goal is to find an input  $x \in \mathcal{X}$  that approximately optimizes  $f$ :

$$\hat{x} \text{ s.t. } f(\hat{x}) \approx \max_x f(x) \quad (1)$$

To solve this optimization problem, we are provided with a static dataset of  $n$  input-output pairs

$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$  collected *offline*, where  $y_i = f(x_i)$ . The optimization algorithm does not have access to objective function  $f$  values on inputs outside the dataset  $\mathcal{D}$ . Hence, this problem is referred to as *offline black-box optimization*.

An offline model-based optimization algorithm produces an input  $\hat{x} \in \mathcal{X}$  which is outside the training dataset  $\mathcal{D}$ . We measure the accuracy of solution in terms of the real objective function value of  $\hat{x}$ , namely  $f(\hat{x})$ . Ideally,  $f(\hat{x})$  should be higher than the best function value seen in the offline dataset  $\mathcal{D}$  (say  $y_{best} = \max\{y_1, y_2, \dots, y_n\}$ ).

## Related Work

Prior work on offline optimization fall into two categories:

**Sampling from Generative Models.** Existing works in this family often focus on learning a generative model of the input space. For example, (Brookes, Park, and Listgarten 2019) and (Fannjiang and Listgarten 2020) employs a variational auto-encoder (Kingma and Welling 2013) which can be conditionally guided by a set of (domain-related) desirable properties. Similarly, (Kumar and Levine 2020) learns an inverse mapping from the performance measure to the input design using conditional generative adversarial network (Mirza and Osindero 2014). However, these approaches require learning a full generative model of the input space in addition to a surrogate model, which adds more complexity to the overall learning process. More recently, (BONET) (Krishnamoorthy, Mashkaria, and Grover 2022a) proposed a sequence modeling based conditional autoregressive model approach that aims to mimic an online black-box optimizer represented by a collection of sorted trajectories synthesized from the offline data. However, BONET requires knowledge of the oracle maxima which makes it unclear whether its performance remains robust if the assumed knowledge of the oracle maxima is not accurate.

Both BONET and our proposed PGS rely on constructing synthetic trajectories from the given offline dataset. However, they differ in the methodology for trajectory synthesis. BONET uses sorted trajectories over the entire offline dataset while PGS generates randomized trajectories from top- $p$  percentile offline data. Unlike BONET, we provide a hyperparameter selection approach to tune  $p$  for any given offline optimization task. Trajectories with monotonically increasing function values may not allow the policy to recover from mistakes, especially as the offline dataset is in a low function-value space. Trajectories with more variability in the function value as done in PGS can overcome this challenge as we demonstrate through ablation studies.

**Gradient-based Updates.** Gradient-based approaches advocate learning a *conservative* proxy that can be directly optimized via gradient ascent to circumvent the erroneous overestimation of the proxy model at out-of-distribution inputs. For example, (Yu et al. 2021a) mitigates the non-smooth nature of neural network using robust model pre-training and model adaptation to ensure a criteria of local smoothness while (Fu and Levine 2021) uses normalized maximum likelihood to handle uncertainty in out-of-distribution (OOD) prediction. Alternatively, (Trabucco

---

**Algorithm 1: Policy-guided Gradient Search (PGS)**

---

**Input:** starting input for search  $x_0 \in \mathcal{D}$ ; surrogate model  $\hat{f}_\theta$ ; policy  $\pi$ ; number of search steps  $T$

**Output:** best uncovered input  $\hat{x}$

```
1: for each search step  $k=1, 2, \dots, T$  do
2:   Predict step size:  $\alpha_{k-1} = \pi(x_{k-1})$ 
3:   Perform gradient step:  $x_k \leftarrow x_{k-1} + \alpha_{k-1} \nabla_x \hat{f}_\theta(x)|_{x=x_{k-1}}$ 
4: end for
5: return the solution of gradient search  $x_T$ 
```

---

et al. 2021) explicitly penalizes high-value prediction for OOD examples to avoid erroneous overestimation of OOD input directly. Nonetheless, the optimization policies of these methods are mostly based on the local gradient information derived from the (imperfect) proxy at each data point, which in general does not always capture the (more global) relationship connecting the distance between two (arbitrary) input coordinates to the difference between their induced objective function values. This motivates us to investigate a reformulation of offline optimization as an offline reinforcement learning task which can naturally encode all the above information within a framework of (learnable) policy-guided gradient search, as detailed in Section .

**Distinction from Learning-to-Optimize Setup:** Our policy learning problem is entirely different than the one considered in a seemingly related literature of learning-to-optimize (L2O) because we are only given a single offline dataset collected from a single task. Existing work on L2O (Andrychowicz et al. 2016) considers the problem setting where multiple datasets belonging to different tasks are available for the learner (aka meta-learning).

### Policy-guided Gradient Search Algorithm

We first provide an overview of the PGS algorithm. Next, we describe a learning approach to create policies for PGS using offline RL. Finally, we outline the offline RL algorithm employed in our implementation and describe a methodology for hyper-parameter selection using only offline data.

**Overview of Policy-guided Gradient Search (PGS).** Our proposed offline model-based optimization approach PGS works as follows. PGS needs a surrogate model  $\hat{f}_\theta : \mathcal{X} \mapsto \mathbb{R}$  to make predictions on inputs outside the offline dataset  $\mathcal{D}$  and a policy  $\pi : \mathcal{X} \mapsto \mathbb{R}^d$  to predict the step size to guide the gradient search towards inputs with high function values. PGS (Algorithm 1) performs  $T$  steps of gradient search using both surrogate model  $\hat{f}_\theta$  and policy  $\pi$  starting from an input  $x_0 \in \mathcal{D}$  with high function value as follows:

$$x_k \leftarrow x_{k-1} + \alpha_{k-1} \nabla_x \hat{f}_\theta(x)|_{x=x_{k-1}} \text{ with } k \in [T] \quad (2)$$

where  $\alpha_{k-1} = \pi(x_{k-1})$  is the step-size predicted by policy  $\pi$ . The solution from the gradient search  $x_T$  is returned as the output. The effectiveness of PGS for offline BBO critically depends on the policy  $\pi$ . We provide an offline RL formulation using the static dataset  $\mathcal{D}$  to create robust policies to improve the accuracy of PGS. This reduction allows

us to leverage the large body of work on offline RL to solve offline BBO problems in a principled manner.

### Policy Learning Formulation

We can train the surrogate function  $\hat{f}_\theta$  from offline dataset  $\mathcal{D}$  using a regression learner. The accuracy of PGS approach for solving offline BBO problem given a model  $\hat{f}_\theta$ , policy  $\pi$ , and starting input  $x_0 \in \mathcal{D}$  is measured in terms of the real objective function value of  $x_T$ , i.e.,  $f(x_T)$ . Therefore, the overall learning objective to create a policy to guide gradient search is as follows.

$$\begin{aligned} \pi^* &\leftarrow \max_{\pi \in \Pi} \mathbb{E}[f(x_T)] \text{ such that} \\ x_0 &\sim \mathcal{X}_{start} \text{ and } x_T = \text{PGS}(x_0, \hat{f}_\theta, \pi, T) \end{aligned} \quad (3)$$

where  $x_0$  is a starting input drawn from the distribution  $\mathcal{X}_{start}$  and  $x_T$  is the solution of PGS (Algorithm 1) with surrogate model  $\hat{f}_\theta$ , policy  $\pi$ , and  $T$  search steps. The optimal  $\pi^*$  from the policy space  $\Pi$  maximizes  $f(x_T)$  in expectation (over  $\mathcal{X}_{start}$ ). Before we discuss the details of our policy learning approach, we explain the corresponding Markov Decision Process (MDP) formulation.

**MDP Definition.** An MDP is a 4-tuple  $\langle S, A, TF, R \rangle$ , where  $S$  is the set of states,  $A$  is the set of actions,  $TF$  is a transition function, and  $R$  is a bounded reward function. A policy  $\pi : S \mapsto A$  is a mapping from states to actions.

**State space  $S$ :** Every candidate input  $x \in \mathcal{X}$  corresponds to one state  $s \in S$ .

**Action space  $A$ :** There are many choices for defining action space. Therefore, we first provide some motivation for the specific choice employed in our formulation. It is well-known in the optimization theory that second-order gradient based algorithms including Newton’s method, Quasi-Newton’s method are much more effective than first-order gradient methods in terms of reaching good solutions faster. The key idea in such methods is to employ second-order gradient (i.e., Hessian matrix) information to precondition the gradient in the solution iterate, i.e.  $x_k = x_{k-1} - B \nabla f(x_{k-1})$  where  $B$  is a preconditioning matrix. For example,  $B$  is the inverse Hessian for Newton’s method. However, as a consequence of requiring second-order gradients (Hessian), most of these approaches are computationally-expensive than first order gradient descent approaches. Inspired by this idea, we parameterize the action space of our gradient search in terms of a diagonal matrix of parameters  $B = \text{diag}(\alpha)$  (i.e., a step-size vector) to strike a good balance between expressiveness and statistical efficiency for policy learning. Using a step-size vector (i.e., one parameter for each input dimension) is more expressive than the typical approach of using a scalar learning rate and more statistically efficient than an entire preconditioning matrix of parameters. This parametrization has also shown to be quite beneficial in the recent literature on learned optimizers or learning to learn methods (Flennerhag et al. 2019; Li and Malik 2016). Hence, we consider each candidate action  $a \in A$  to be a step-size vector  $\alpha$ . Note that each parameter value in  $\alpha$  could be either positive or negative.

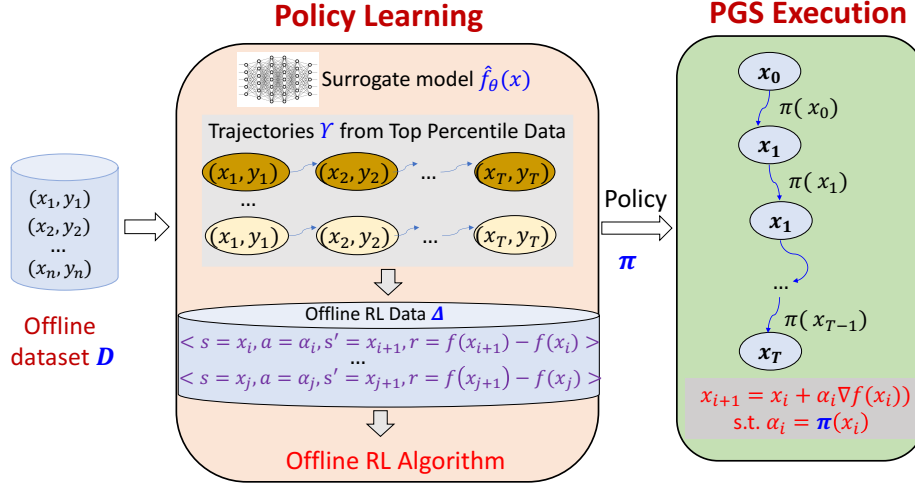


Figure 1: High-level overview of policy-guided gradient search approach for offline black-box optimization (BBO). The key idea is to cast the offline BBO problem as an offline RL problem. This reduction is accomplished by constructing random trajectories from a subset of inputs with high function values from the given offline data  $\mathcal{D}$  (say Top  $p$  percentile which is determined in a data-driven manner using our offline state estimation approach). The policy  $\pi$  corresponds to selecting a step-size vector for a gradient based update on a trained surrogate model  $\hat{f}_\theta$ . Given a learned policy  $\pi$ , surrogate model  $\hat{f}_\theta$  and a starting input  $x_0$  with high function value sampled from the offline data  $\mathcal{D}$ , PGS performs  $T$  steps of gradient search by asking the policy  $\pi$  to predict the step-size vector  $\alpha$  at each search step.

**Transition function  $TF$ :** The transition function  $TF : S \times A \mapsto S$  is deterministic in our formulation. It takes a state  $s = x$  and an action  $a = \alpha$  to produce the next state  $s' = x'$ .

$$x' \leftarrow x + \alpha \nabla_x \hat{f}_\theta(x) \quad (4)$$

**Reward function  $R$ :** The reward function  $R : S \times A \mapsto \mathbb{R}$  is defined as follows. The reward  $R(s, a)$  of taking an action  $a = \alpha$  at a state  $s = x$  is equal to  $f(x') - f(x)$ , where  $s' = x'$  is the next state. In other words, reward is the difference between objective function values of next state and current state. It is positive when we go from states with low function values to states with high function values and vice versa.

The above MDP construction is a deliberate design choice in our reduction approach to create policies using offline RL methods. In fact, improved offline optimization performance from PGS using this simple MDP in terms of state and reward representation demonstrates its effectiveness. We believe that considering more complex choices in the MDP (e.g., sequence modeling based unsupervised state embeddings and/or reward engineering) can be an interesting avenue for future research.

### Trajectory Construction and RL Approach

Recall that we only have access to the function values for a subset of inputs from  $\mathcal{X}$  as part of the offline dataset  $\mathcal{D}$ . We can employ  $\mathcal{D}$  to create a set of random trajectories over the Top  $p$  percentile data (sequence of inputs whose objective function values are high) and employ standard RL algorithms to create a policy  $\pi$  for policy-guided search. We describe the two key steps of this approach below.

**Trajectory synthesis.** We construct the set of trajectories from a prioritized subset  $\mathcal{D}_{top}$  of inputs with high function values from the given offline dataset  $\mathcal{D}$ . This choice is primarily motivated by the need to align our training process with the test-time search algorithm. Recall that during the test-time, we perform  $T$  gradient search steps guided by the trained policy starting from inputs with high objective function values. Naturally, it is useful to induce the policy in search regions from inputs with high function values as well. Therefore, we consider a simple and scalable choice of picking the examples lying in the  $p^{th}$ -percentile of the given offline dataset  $\mathcal{D}$  sorted based on their objective function values. Note that our approach does not use any data outside the given offline dataset  $\mathcal{D}$ .

Restricting the trajectory synthesis to the subset  $\mathcal{D}_{top}$  also allows more flexibility and robustness in constructing the trajectories for offline RL. We consider a simple approach to construct trajectories from  $\mathcal{D}_{top}$ . To create a trajectory of length  $T$ ,  $\tau = (x_1, y_1), (x_2, y_2), \dots, (x_T, y_T)$ , we randomly sample a sequence of input-output pairs from  $\mathcal{D}_{top}$  without replacement. We create a collection of trajectories  $\Upsilon = \{\tau_1, \tau_2, \dots, \tau_m\}$  by repeating this procedure several times. However, the decision to select the prioritized subset  $\mathcal{D}_{top}$  has an inherent trade-off as this may lead to loss of diversity in the trajectories when we discard inputs from the low-ranked objective function values. Importantly, the right trade-off to achieve improved offline optimization performance may vary from one benchmark to another. Therefore, we provide a principled hyper-parameter selection methodology based on offline state estimation (explained later) to select the value of  $p$  for Top  $p$  percentile which can be potentially useful for other offline optimization approaches.

---

**Algorithm 2: Learning to Guide Gradient Search**

---

**Input:** offline dataset  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ ; number of input dimensions  $d$ ; number of search steps  $T$

**Output:** model  $\hat{f}_\theta$  and policy  $\pi$

- 1: Learn surrogate model  $\hat{f}_\theta$ :  $\hat{f}_\theta \leftarrow \text{REGRESSION-LEARNER}(\mathcal{D})$
  - 2: Select  $\mathcal{D}_{top} \subseteq \mathcal{D}$  consisting of inputs with Top  $p$  percentile highest objective values
  - 3: Create a collection of random trajectories of length  $T$  using  $\mathcal{D}_{top}$ :  $\Upsilon = \{\tau_1, \tau_2, \dots, \tau_m\}$  s.t every trajectory  $\tau_i$  consists of a random sequence of examples from the Top  $p$  percentile data
  - 4: Training data for policy learning  $\Delta \leftarrow \emptyset$
  - 5: **for** each trajectory  $\tau = (x_1, y_1), \dots, (x_T, y_T) \in \Upsilon$  **do**
  - 6:   **for** each pair  $(x_{k-1}, y_{k-1})$  and  $(x_k, y_k)$  from  $\tau$  **do**
  - 7:     Add a sample  $(s, a, s', r)$  to  $\Delta$   
      where current state  $s = x_{k-1}$ , next state  $s' = x_k$ , reward  $r = y_k - y_{k-1}$ , and action  $a = \alpha_{k-1}$  s.t  $x_k \leftarrow x_{k-1} + \alpha_{k-1} \nabla_x \hat{f}_\theta(x)|_{x=x_{k-1}}$
  - 8:   **end for**
  - 9: **end for**
  - 10: Learn policy  $\pi$ :  
     $\pi \leftarrow \text{OFFLINE-REINFORCEMENT-LEARNER}(\Delta)$
  - 11: **return** the learned surrogate function  $\hat{f}_\theta$  and policy  $\pi$
- 

**Baseline RL approach.** Given the set of trajectories  $\Upsilon = \{\tau_1, \tau_2, \dots, \tau_m\}$ , a straightforward approach would be to use a standard RL algorithm (Sutton and Barto 2018) with a function approximator (Mnih et al. 2015) to find a policy  $\pi$ . Although this approach is very simple, it will most likely fail as the learned policy  $\pi$  cannot handle out-of-distribution (OOD) states/inputs, i.e., inputs outside the offline dataset  $\mathcal{D}$ . Instead, we propose using an offline RL approach which leverages explicit conservatism to find policies which are robust to OOD data. Indeed, our experiments show that incorporating OOD robustness via offline RL is critical for good performance on offline BBO.

### Offline RL Approach for Policy Learning

Prior work on offline RL has shown significant successes in learning policies from a given offline dataset of experiences (i.e., 4-tuples consisting of state, action, next state, and reward). Inspired by these successes, we consider an offline RL approach to learn policies that are robust to OOD inputs outside the offline data  $\mathcal{D}$ . In our MDP formulation, we do not know the reward for some pairs of states, when we do not know the real objective function value of the corresponding input of at least one of the states.

To learn an effective policy to handle OOD states via offline RL, an agent should deviate from the available behavior in the logged/offline data. However, distributional shift scenarios are overestimated by the value function, leading to mediocre policies. Model-free approaches circumvent such bias by embedding constraints over the objective function via some form of regularization. Constraints include the policy, the value function, or both (Fujimoto et al. 2019; Peng et al. 2019; Kumar et al. 2019; Wu, Tucker, and Nachum 2020; Kumar et al. 2020; Fujimoto and Gu 2021). Model-based approaches create conservative MDPs of the environment and proceed to apply regular online RL methods since

they have access to the environment dynamics (Kidambi et al. 2020; Yu et al. 2020, 2021b).

Our offline RL approach (see Algorithm 2) shares the first step of the above-mentioned baseline RL approach, namely, trajectory synthesis. We employ the dataset of random trajectories  $\Upsilon = \{\tau_1, \tau_2, \dots, \tau_m\}$  to create a training set which is appropriate for offline RL as explained below.

**Reduction to offline RL.** Recall that each training example for offline RL is a 4-tuple of the form  $(s, a, s', r)$ , where  $s$  is a current state,  $a$  is an action,  $s'$  is the next state resulting from taking an action  $a$  at state  $s$ ,  $r$  is the reward for going to next state  $s'$  from  $s$ . We generate a set of training examples of this form using random trajectories as follows (Line 5-9 in Algorithm 2). For each trajectory  $\tau \in \Upsilon = (x_1, y_1), (x_2, y_2), \dots, (x_T, y_T)$ , we create one offline RL training example for every pair of inputs  $(x_{k-1}, y_{k-1})$  and  $(x_k, y_k)$  from  $\tau$ . The state  $s = x_{k-1}$ , the action  $a = \alpha$  such that the gradient step will result in the next state  $s' = x_k$ :  $x_k \leftarrow x_{k-1} + \alpha \nabla_x \hat{f}_\theta(x)|_{x=x_{k-1}}$ , the next state  $s' = x_k$ , and the reward  $r = f(x_k) - f(x_{k-1})$ . The aggregate set of offline RL training examples  $\Delta$  over all trajectories in  $\Upsilon$  are given to an offline RL algorithm to create the policy  $\pi$  (Line 10 in Algorithm 2). The main advantage of this reduction is that it allows us to leverage prior work on offline RL to solve the challenging problem of offline black-box optimization.

**Conservative Q-learning (CQL).** To implement our PGS approach, we can employ any existing offline RL algorithm. We chose conservative Q-learning because of its demonstrated effectiveness in practice (Kumar et al. 2020). For the sake of completeness, we briefly describe the CQL approach and how it is configured for our experimental evaluation. To address distributional shift, CQL learns a lower bound of the true policy value leading to a more reliable value function estimate. Along with the standard temporal difference error, a regularizer is added to minimize Q-values for unseen actions sampled from a distribution  $\mu$  – see Eq. (5), which optimizes for  $\hat{Q}^\pi = \arg \min_Q \ell(Q)$  where

$$\begin{aligned} \ell(Q) \triangleq & \max_{\mu(a|s)} \left( \mathbb{E}_{s \sim \mathcal{D}} \mathbb{E}_{a \sim \mu} [Q(s, a)] - \mathbb{E}_{(s, a) \sim \mathcal{D}} [Q(s, a)] \right) \\ & + \frac{1}{2\beta} \mathbb{E}_{(s, a, s') \sim \mathcal{D}} [\Delta(s, a, s')] \end{aligned} \quad (5)$$

with  $\Delta(s, a, s') = (r(s, a) + \gamma \mathbb{E}_\pi [Q(s', a')] - Q(s, a))^2$

The conservative Q function is then used to train a soft-actor critic agent in our case (Haarnoja et al. 2018). Since our MDP is deterministic in nature, CQL’s theoretical analysis is also directly applicable to the offline optimization setting. This is another advantage of our reduction formulation!

### Offline State Estimation via Latent Embeddings for Hyperparameter Selection

We propose a principled methodology for selecting the key hyper-parameters of our PGS approach. The main idea is to assign estimated function values to unknown inputs (outside the offline dataset) such that they are mostly correlated

with the true function values. We leverage offline state representation learning objectives (Yang and Nachum 2021; Oh, Singh, and Lee 2017) to embed the inputs (states in our MDP formulation) into a latent embedding space. This idea of pre-trained state representations from offline data has been shown to be quite effective for RL algorithms (Yang and Nachum 2021). To get an offline estimate of the performance of inputs suggested by PGS, we employ the following algorithmic procedure which we refer to as *Offline State Estimation via Latent embeddings (OSEL)*:

- Embed unknown input into the latent embedding space.
- Get a  $K$ -nearest neighbor regressor based estimate of the performance, where we pick  $K$  nearest offline dataset inputs in the latent space as neighbors.

We would like to emphasize that the OSEL procedure only uses the offline dataset  $\mathcal{D}$  and does not rely on external sources. We perform pre-training of embeddings using random trajectories generated from the entire offline dataset. Our hyper-parameter selection methodology represents a thoughtful and principled means of improving the quality of input representations, which, in turn, contributes to the overall performance of our approach.

## Experiments and Results

We first describe our setup including the benchmarks, evaluation methodology, and baselines. Next, we discuss the results comparing the PGS approach with baseline methods.

### Experimental Setup

**Benchmark tasks.** We employ six challenging benchmark tasks (and corresponding datasets) from diverse domains. All these datasets and the oracle evaluations are accessed via the design-bench benchmark (Trabucco et al. 2022).

**1) D’Kitty Morphology:** This task requires optimizing the morphology of a four-legged robot named D’Kitty in order to reach a given location (Ahn et al. 2020). The morphological parameters of the robot is defined over a *56-dimensional continuous* search space. **2) Ant Morphology:** Similar to the D’Kitty benchmark, this task (Brockman et al. 2016) requires optimizing the morphology of a 3D robot. The search space of parameters is continuous and 60-dimensional in size. **3) Superconductor:** This task (Brookes, Park, and Listgarten 2019) involves designing superconductors with high critical temperatures which has many engineering and material science applications and each input point is represented by a 86-dimensional continuous vector. **4) GFP:** The goal of this task (Rao et al. 2019) is to find fluorescence maximizing proteins. Each point in the search space denotes a protein which is represented by a 237-dimensional vector. Each input dimension can take values from 20 different amino acids. **5) TF Bind8:** This task aims to maximize the binding activity score between a given human transcription factor and a DNA sequence. It’s a discrete task with 8-length DNA sequences represented as vectors with values from 4 categories. **6) UTR:** This task (Sample et al. 2019) requires optimizing a search space of 50-dimensional DNA sequences to maximize the expression level of a given gene.

**Configuration of algorithms and baselines.** We compare PGS against baselines including COMs (Trabucco et al. 2021), NEMO (Fu and Levine 2021), ROMA (Yu et al. 2021a), BDI (Chen et al. 2022), BONET (Krishnamoorthy, Mashkaria, and Grover 2022b) and other baselines from design-bench (Trabucco et al. 2022). We took the results for all baselines from their respective papers. Since NEMO and ROMA don’t report normalized scores, and their original papers lack a few tasks, we take their results from the state-of-the-art BDI paper (Chen et al. 2022). This is reasonable because all baselines use the same design-bench benchmark and evaluation methodology. We note that BONET utilizes double the evaluation budget (i.e., 256 points) compared to all other methods. Therefore, for fair evaluation with all the baselines, we ran BONET code (from official implementation <https://github.com/siddarthk97/bonet>) to generate 128 points for evaluation.

We configure PGS as follows. For each task, we normalize inputs and outputs before we train a vanilla multi-layer perceptron,  $\hat{f}_\theta$ , with two hidden layers with 2048 units and ReLU activation.  $\hat{f}_\theta$  is trained to minimize the mean squared error of function values. We employ the publicly available implementation of CQL <https://github.com/young-geng/CQL>. PGS is based on gradient updates over continuous valued inputs in contrast to discrete tasks. To mitigate this, we learn a latent representation of the discrete inputs. We apply a variational autoencoder and train PGS over the learned latent space. Note that we decode the results before oracle evaluation.

We configure OSEL for hyper-parameter selection as follows. We employ contrastive predictive loss objective as used in value prediction networks (VPNs) (Oh, Singh, and Lee 2017) where the key idea involves learning to predict  $m$ -step future rewards and value functions starting from a given state. We used the publicly available implementation of VPns ([https://github.com/google-research/google-research/tree/master/rl\\_repr](https://github.com/google-research/google-research/tree/master/rl_repr)). We employ 20000 trajectories of length  $T=50$  to align training with test-time search process. We evaluate four different values of  $p = \{10, 20, 30, 40\}$  for Top  $p$  percentile data and number of epochs of CQL ranging from 50 to 400 in increments of 50 and picked the configuration with the best OSEL performance.

**Evaluation methodology.** We follow the methodology as introduced in the design-bench benchmark (Trabucco et al. 2022) and employed by all prior work on offline BBO. Each algorithm generates a set of  $N$  points which are evaluated by oracle and the 100th percentile (best value among the  $N$  points) is computed to compare the different approaches. All objective/oracle values are normalized to  $[0, 1]$  by using the mean and max from a larger unobserved data set. We run PGS and BONET on each task for five different runs and report the mean and standard deviation in the results section.

### Results and Discussion

**PGS w/ offline RL vs. standard RL.** To clearly show that the offline RL component of PGS approach brings significant advantage, we perform ablation by replacing it with a standard RL algorithm while keeping everything else same

Continuous Tasks	Superconductor	Ant Morphology	D’Kitty Morphology
<b>PGS w/ Offline RL</b>	<b>0.563 <math>\pm</math> 0.058</b>	<b>0.949 <math>\pm</math> 0.017</b>	<b>0.966 <math>\pm</math> 0.013</b>
PGS w/ Standard RL	0.528 $\pm$ 0.012	0.303 $\pm$ 0.017	0.900 $\pm$ 0.007
Discrete Tasks	GFP	TF Bind 8	UTR
<b>PGS w/ Offline RL</b>	<b>0.864 <math>\pm</math> 0.000</b>	<b>0.981 <math>\pm</math> 0.015</b>	<b>0.713 <math>\pm</math> 0.009</b>
PGS w/ Standard RL	<b>0.864 <math>\pm</math> 0.000</b>	0.774 $\pm$ 0.063	0.686 $\pm$ 0.013

Table 1: Ablation Results showing the ablation of using standard RL as opposed to offline RL to create the policy for PGS.

Method	Superconductor	Ant Morphology	D’Kitty Morphology
$\mathcal{D}(\text{best})$	0.399	0.565	0.884
BO-qEI	0.402 $\pm$ 0.034	0.819 $\pm$ 0.000	0.896 $\pm$ 0.000
CMA-ES	0.465 $\pm$ 0.024	<b>1.214 <math>\pm</math> 0.732</b>	0.724 $\pm$ 0.001
REINFORCE	0.481 $\pm$ 0.013	0.266 $\pm$ 0.032	0.562 $\pm$ 0.196
CbAS	0.503 $\pm$ 0.069	0.876 $\pm$ 0.031	0.892 $\pm$ 0.008
Auto.CbAS	0.421 $\pm$ 0.045	0.882 $\pm$ 0.045	0.906 $\pm$ 0.006
MIN	0.469 $\pm$ 0.023	0.913 $\pm$ 0.036	0.945 $\pm$ 0.012
BONET	0.411 $\pm$ 0.024	0.927 $\pm$ 0.010	0.954 $\pm$ 0.009
Grad	0.518 $\pm$ 0.024	0.293 $\pm$ 0.023	0.874 $\pm$ 0.022
COMs	0.439 $\pm$ 0.033	0.944 $\pm$ 0.016	0.949 $\pm$ 0.015
ROMA	0.476 $\pm$ 0.024	0.814 $\pm$ 0.051	0.905 $\pm$ 0.018
NEMO	0.488 $\pm$ 0.034	0.814 $\pm$ 0.043	0.924 $\pm$ 0.012
BDI	0.520 $\pm$ 0.005	0.962 $\pm$ 0.000	0.941 $\pm$ 0.000
<b>PGS (ours)</b>	<b>0.563 <math>\pm</math> 0.058</b>	0.949 $\pm$ 0.017	<b>0.966 <math>\pm</math> 0.013</b>

Table 2: Results comparing PGS and baseline methods on benchmark tasks with continuous search spaces. PGS finds better solutions than all other approaches on Superconductor and D’Kitty Morphology and is competitive on Ant Morphology.

Method	GFP	TF Bind 8	UTR	Mean Rank
$\mathcal{D}(\text{best})$	0.789	0.439	0.593	
BO-qEI	0.254 $\pm$ 0.352	0.798 $\pm$ 0.083	0.684 $\pm$ 0.000	11.33/13
CMA-ES	0.054 $\pm$ 0.002	0.953 $\pm$ 0.022	0.707 $\pm$ 0.014	7/13
REINFORCE	<b>0.865 <math>\pm</math> 0.000</b>	0.948 $\pm$ 0.028	0.688 $\pm$ 0.010	8.16/13
CbAS	<b>0.865 <math>\pm</math> 0.000</b>	0.927 $\pm$ 0.051	0.694 $\pm$ 0.010	6.33/13
Auto.CbAS	<b>0.865 <math>\pm</math> 0.000</b>	0.910 $\pm$ 0.044	0.691 $\pm$ 0.012	7.5/13
MIN	<b>0.865 <math>\pm</math> 0.001</b>	0.905 $\pm$ 0.052	0.697 $\pm$ 0.010	5.83/13
BONET	<b>0.864 <math>\pm</math> 0.000</b>	0.911 $\pm$ 0.034	0.688 $\pm$ 0.011	7.33/13
Grad	<b>0.864 <math>\pm</math> 0.001</b>	0.977 $\pm$ 0.025	0.695 $\pm$ 0.013	6.5/13
COMs	<b>0.864 <math>\pm</math> 0.000</b>	0.945 $\pm$ 0.033	0.699 $\pm$ 0.011	5.33/13
ROMA	0.558 $\pm$ 0.395	0.928 $\pm$ 0.038	0.690 $\pm$ 0.012	8.66/13
NEMO	0.150 $\pm$ 0.270	0.905 $\pm$ 0.048	0.694 $\pm$ 0.015	8.5/13
BDI	<b>0.864 <math>\pm</math> 0.000</b>	0.973 $\pm$ 0.000	<b>0.760 <math>\pm</math> 0.000</b>	3/13
<b>PGS (ours)</b>	<b>0.864 <math>\pm</math> 0.000</b>	<b>0.981 <math>\pm</math> 0.015</b>	0.713 $\pm$ 0.009	<b>2.16/13</b>

Table 3: Results comparing PGS and baselines on benchmarks with discrete input spaces. PGS finds better solution than all other approaches on TF Bind 8, reaches the best benchmarks solution on GFP and is competitive on UTR. The last column show the mean rank computed across all the 6 tasks to measure the overall effectiveness of methods for multiple tasks.

including the synthesized trajectories. We chose soft-actor critic as the RL method. This allows us to keep it consistent with our offline RL method (CQL) since it builds upon soft-actor critic as one of its components. Results in Table 1 show that the performance of PGS becomes consistently worse by using standard RL over CQL, which demonstrates the importance of employing offline RL algorithm to correct for OOD data using the principle of conservatism.

**Comparison with state-of-the-art.** We present the results comparing PGS with all baselines in Tables 2 (continuous input spaces) and 3 (discrete input spaces). Each column with the task name shows the 100th percentile (top score) found by each method on the corresponding task. The  $\mathcal{D}(\text{best})$  row refers to the highest score in each task’s offline data. We also show the mean rank achieved by the methods computed across all the tasks. We highlight in bold the methods with the best results for every task.

Task	AntMorphology	DKittyMorphology	Superconductor	TFBind8
PGS (IQL)	$0.939 \pm 0.021$	$0.959 \pm 0.002$	$0.532 \pm 0.000$	$0.960 \pm 0.027$
PGS (CQL)	$0.949 \pm 0.017$	$0.966 \pm 0.013$	$0.563 \pm 0.058$	$0.981 \pm 0.015$

Table 4: Table comparing the results of training with another offline RL algorithm: IQL (Kostrikov, Nair, and Levine 2022)

Task	AntMorphology	DKittyMorphology	Superconductor	TFBind8
PGS (entire data)	$0.890 \pm 0.022$	$0.947 \pm 0.013$	$0.533 \pm 0.036$	$0.955 \pm 0.024$
PGS (top $p$ )	<b><math>0.949 \pm 0.017</math></b>	<b><math>0.966 \pm 0.013</math></b>	<b><math>0.563 \pm 0.058</math></b>	<b><math>0.981 \pm 0.015</math></b>

Table 5: Table comparing the results of PGS using trajectories synthesized with top  $p$  percentile data vs. entire offline dataset.

Task	AntMorphology	DKittyMorphology	TFBind8
PGS (monotonic)	$0.916 \pm 0.027$	$0.964 \pm 0.013$	$0.925 \pm 0.031$
PGS (top $p$ )	<b><math>0.949 \pm 0.017</math></b>	<b><math>0.966 \pm 0.013</math></b>	<b><math>0.981 \pm 0.015</math></b>

Table 6: Table comparing the results of PGS using top  $p$  random trajectories vs. monotonically increasing trajectories

Cumulatively, PGS achieves an average ranking of 2.16 across all tasks, which is higher than all the baselines. Individually, PGS finds the best scores on 4 (Superconductor, D’Kitty Morphology, TFBind8, and GFP) out of the 6 tasks and very close to the best baseline on Ant Morphology and UTR. PGS significantly outperforms baselines, especially in Superconductor and D’Kitty tasks, and excels in surpassing the best designs from offline data across all tasks, showcasing its effectiveness in offline BBO problem-solving.

Since the performance of offline RL critically depends on the amount of state space coverage in the offline dataset, one potential limitation of PGS is that it may not perform well if the coverage is bad. For example, the offline dataset for the Ant morphology task are picked from the lowest scoring parts of the objective space relative to other benchmarks.

**Ablation experiments.** We conducted several ablations to demonstrate the effectiveness and generality of PGS.

- *PGS with other offline RL methods:* We conducted ablation experiments on PGS by replacing CQL with another off-the-shelf offline RL algorithm, implicit Q learning (IQL) (Kostrikov, Nair, and Levine 2022). Remarkably, we observed consistent and strong performance across various tasks. These results in Table 4, averaged over five runs, underscore the robustness of PGS’s core concept: the reduction to an offline RL problem.
- *PGS with Top- $p$  percentile vs. entire offline data:* We run the ablation of running offline RL on trajectories constructed from top  $p$  prioritized data subset versus that from the entire offline dataset. The results shown in Table 5 empirically confirm that the top  $p$  percentile subset selection is a better strategy for trajectory construction.
- *PGS with monotonic trajectories over entire offline data:* We perform PGS ablation to compare our proposed trajectory synthesis approach (random trajectories over the top  $p$  percentile offline data) with BONET (monotonic trajectories over the entire offline data). The results

shown in Table 6 provide empirical evidence that our trajectory design approach reaches better solutions.

- *Other ablations:* We also performed ablation experiments on variations in number of gradient search steps at test-time and examining the performance of PGS with varying dataset sizes for offline RL. Furthermore, We analyzed the policy-guided search trajectory at test-time by computing the norms of action vectors. We observed that the norms of action sequence consistently decrease as the number of search steps increase (aggressive to conservative exploration of search space). This phenomenon suggests that our learned policy becomes more adept at guiding the search as we approach regions associated with high-value designs. For more details on all ablation experiments, please refer to the Appendix.

## Summary and Future Work

This paper introduces the new perspective of policy-guided gradient search for offline black-box optimization (BBO). This perspective is aimed at improving the search strategy in offline BBO, which complements prior methods that have focused on improving surrogate models while using fixed search strategies. Empirical results show that learned search strategies can help in improving the accuracy of optimization significantly across multiple benchmarks. There are many avenues of future work. First, exploring more sophisticated trajectory sampling methods will improve the effectiveness of many offline optimization methods including PGS and BONET. Second, while our hyperparameter selection method based on offline state estimation showed promise, but finding the appropriate hyper-parameters for offline optimization remains an open and important research challenge. Finally, identifying suitable offline optimization methods for specific problems based on problem properties is a valuable future research direction which can benefit practitioners and motivate researchers to drive algorithmic advancements to improve the Pareto front of solutions.



## Acknowledgements

The authors gratefully acknowledge the in part support from National Science Foundation (NSF) grants IIS-1845922 and CNS-2308530. The views expressed are those of the authors and do not reflect the official policy or position of the NSF.

## References

- Ahn, M.; Zhu, H.; Hartikainen, K.; Ponte, H.; Gupta, A.; Levine, S.; and Kumar, V. 2020. Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on robot learning*, 1300–1313. PMLR.
- Andrychowicz, M.; Denil, M.; Gomez, S.; Hoffman, M. W.; Pfau, D.; Schaul, T.; Shillingford, B.; and De Freitas, N. 2016. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29.
- Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.
- Brookes, D. H.; Park, H.; and Listgarten, J. 2019. Conditioning by adaptive sample2019ling for robust design. *CoRR*, abs/1901.10060.
- Chen, C.; Zhang, Y.; Fu, J.; Liu, X.; and Coates, M. 2022. Bidirectional Learning for Offline Infinite-width Model-based Optimization. *CoRR*, abs/2209.07507.
- Daumé, H.; Langford, J.; and Marcu, D. 2009. Search-based structured prediction. *Machine learning*, 75: 297–325.
- Deshwal, A.; and Doppa, J. R. 2021. Combining Latent Space and Structured Kernels for Bayesian Optimization over Combinatorial Spaces. In *NeurIPS*, 8185–8200.
- Deshwal, A.; Jayakodi, N. K.; Joardar, B. K.; Doppa, J. R.; and Pande, P. P. 2019. MOOS: A Multi-Objective Design Space Exploration and Optimization Framework for NoC enabled Manycore Systems. *ACM TECS*.
- Deshwal, A.; Simon, C. M.; and Doppa, J. R. 2021. Bayesian optimization of nanoporous materials. *Molecular Systems Design & Engineering*, 6(12): 1066–1086.
- Doppa, J. R.; Fern, A.; and Tadepalli, P. 2014. HC-Search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research*, 50: 369–407.
- Eriksson, D.; Pearce, M.; Gardner, J.; Turner, R. D.; and Poloczec, M. 2019. Scalable global optimization via local Bayesian optimization. *Advances in neural information processing systems*, 32.
- Fannjiang, C.; and Listgarten, J. 2020. Autofocused oracles for model-based design. *CoRR*, abs/2006.08052.
- Flennerhag, S.; Rusu, A. A.; Pascanu, R.; Visin, F.; Yin, H.; and Hadsell, R. 2019. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*.
- Fu, J.; and Levine, S. 2021. Offline Model-Based Optimization via Normalized Maximum Likelihood Estimation. *CoRR*, abs/2102.07970.
- Fujimoto, S.; Conti, E.; Ghavamzadeh, M.; and Pineau, J. 2019. Benchmarking Batch Deep Reinforcement Learning Algorithms. *arXiv preprint arXiv:1910.01708*.
- Fujimoto, S.; and Gu, S. S. 2021. A Minimalist Approach to Offline Reinforcement Learning. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 20132–20145.
- Gao, W.; Mahajan, S. P.; Sulam, J.; and Gray, J. J. 2020. Deep learning in protein structural modeling and design. *Patterns*, 1(9).
- Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 1856–1865. PMLR.
- Higgins, I.; Matthey, L.; Pal, A.; Burgess, C.; Glorot, X.; Botvinick, M.; Mohamed, S.; and Lerchner, A. 2016. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*.
- Huang, G.; Hu, J.; He, Y.; Liu, J.; Ma, M.; Shen, Z.; Wu, J.; Xu, Y.; Zhang, H.; Zhong, K.; et al. 2021. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems*, 26(5): 1–46.
- Kidambi, R.; Rajeswaran, A.; Netrapalli, P.; and Joachims, T. 2020. MOREL: Model-Based Offline Reinforcement Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.
- Kingma, D. P.; and Welling, M. 2013. Auto-Encoding Variational Bayes.
- Kostrikov, I.; Nair, A.; and Levine, S. 2022. Offline Reinforcement Learning with Implicit Q-Learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.
- Krishnamoorthy, S.; Mashkaria, S. M.; and Grover, A. 2022a. Generative Pretraining for Black-Box Optimization. *arXiv:2206.10786*.
- Krishnamoorthy, S.; Mashkaria, S. M.; and Grover, A. 2022b. Generative pretraining for black-box optimization. *arXiv preprint arXiv:2206.10786*.
- Kumar, A.; Fu, J.; Soh, M.; Tucker, G.; and Levine, S. 2019. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d’Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 11761–11771.

- Kumar, A.; and Levine, S. 2020. Model inversion networks for model-based optimization. *Advances in Neural Information Processing Systems*, 33: 5126–5137.
- Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative Q-Learning for Offline Reinforcement Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.
- Li, K.; and Malik, J. 2016. Learning to optimize. *arXiv preprint arXiv:1606.01885*.
- Mirza, M.; and Osindero, S. 2014. Conditional Generative Adversarial Nets. *CoRR*, abs/1411.1784.
- Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.
- Oh, J.; Singh, S.; and Lee, H. 2017. Value prediction network. *Advances in neural information processing systems*, 30.
- Peng, X. B.; Kumar, A.; Zhang, G.; and Levine, S. 2019. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. *CoRR*, abs/1910.00177.
- Rao, R.; Bhattacharya, N.; Thomas, N.; Duan, Y.; Chen, P.; Canny, J.; Abbeel, P.; and Song, Y. 2019. Evaluating protein transfer learning with TAPE. *Advances in neural information processing systems*, 32.
- Sample, P. J.; Wang, B.; Reid, D. W.; Presnyak, V.; McFadyen, I. J.; Morris, D. R.; and Seelig, G. 2019. Human 5 UTR design and variant effect prediction from a massively parallel translation assay. *Nature biotechnology*, 37(7): 803–809.
- Schneider, P.; Walters, W. P.; Plowright, A. T.; Sieroka, N.; Listgarten, J.; Goodnow Jr, R. A.; Fisher, J.; Jansen, J. M.; Duca, J. S.; Rush, T. S.; et al. 2020. Rethinking drug design in the artificial intelligence era. *Nature Reviews Drug Discovery*, 19(5): 353–364.
- Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and De Freitas, N. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175.
- Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.
- Trabucco, B.; Geng, X.; Kumar, A.; and Levine, S. 2022. Design-Bench: Benchmarks for Data-Driven Offline Model-Based Optimization.
- Trabucco, B.; Kumar, A.; Geng, X.; and Levine, S. 2021. Conservative Objective Models for Effective Offline Model-Based Optimization. *CoRR*, abs/2107.06882.
- Wang, H.; Fu, T.; Du, Y.; Gao, W.; Huang, K.; Liu, Z.; Chandak, P.; Liu, S.; Van Katwyk, P.; Deac, A.; et al. 2023. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972): 47–60.
- Wu, Y.; Tucker, G.; and Nachum, O. 2020. Behavior Regularized Offline Reinforcement Learning.
- Yang, M.; and Nachum, O. 2021. Representation matters: offline pretraining for sequential decision making. In *International Conference on Machine Learning*, 11784–11794. PMLR.
- Yu, S.; Ahn, S.; Song, L.; and Shin, J. 2021a. RoMA: Robust Model Adaptation for Offline Model-based Optimization. *CoRR*, abs/2110.14188.
- Yu, T.; Kumar, A.; Rafailov, R.; Rajeswaran, A.; Levine, S.; and Finn, C. 2021b. COMBO: Conservative Offline Model-Based Policy Optimization. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 28954–28967.
- Yu, T.; Thomas, G.; Yu, L.; Ermon, S.; Zou, J. Y.; Levine, S.; Finn, C.; and Ma, T. 2020. MOPO: Model-based Offline Policy Optimization. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

## A Appendix

In this section, we provide more details about the experimental setup and include ablation results to further evaluate our PGS approach. The hyper-parameters for learning the surrogate model and OSEL embeddings are given in Table 7. They are kept same as the previous related work and are fairly robust.

### A.1 Hyper-parameter selection via OSEL

In this section, we describe the step-by-step procedure involved in selecting two key hyperparameters of PGS: top  $p$  percentile of the data for trajectory construction and number of epochs required for training the offline RL policy. Here, the number of epochs are conditionally dependent on the choice of  $p$  hyper-parameter. For each candidate  $p$  value in  $\{10, 20, 30, 40\}$ , we execute following steps: i) Generate a set of trajectories, ii) Train the offline RL (CQL) algorithm for 400 epochs while logging the agent at every 50 epochs resulting in a total of 8 policies, iii) Perform multiple (128) policy guided searches with each candidate logged agent which is exactly similar to the test-time evaluation, iv) Apply OSEL to evaluate the final designs suggested by policy guided search for each agent and compute their average as the final score. This procedure allows us to compute the score for each element in the cartesian product of  $p = \{10, 20, 30, 40\}$  and no of epochs  $\{50, 100, 150, 200, 250, 300, 350, 400\}$ . We pick the hyper-parameter pair with the best average OSEL score. The values chosen by our hyper-parameter selection procedure for each task are reported in Tables 8 and 9.

### A.2 Discrete Tasks VAE details

We employ a  $\beta$ -VAE (Higgins et al. 2016) framework with  $\beta = 1.0$  to train an encoder and a decoder network. Both the encoder and decoder networks consist of four residual convolution blocks. Each block comprises two 1D convolution layers with a hidden size of 64 and a kernel size of 3. We use the publicly available code of  $\beta$ -VAE, from the design baselines, and train for 50 epochs.

### A.3 Ablation experiments and results

As mentioned in the main text, we present additional ablation results here.

**PGS test-time action norms** We examined the norms of actions generated by the offline RL algorithm, as depicted in figure 2, per the number of search steps. Clearly, the action norms decrease as the search progresses. We hypothesize that this trend arises from the offline RL algorithm’s improved guidance of the search process, encouraging more conservative actions as we approach regions near the optimal design.

**Test-time trajectory length (number of gradient search steps)** We ablate the number of search steps policy guided search is run during test-time with a policy trained for 50 search steps and show the results in Table 10. Recall that we employ this search starting from 128 different points generating a set of 128 points for evaluation. Based on the results,

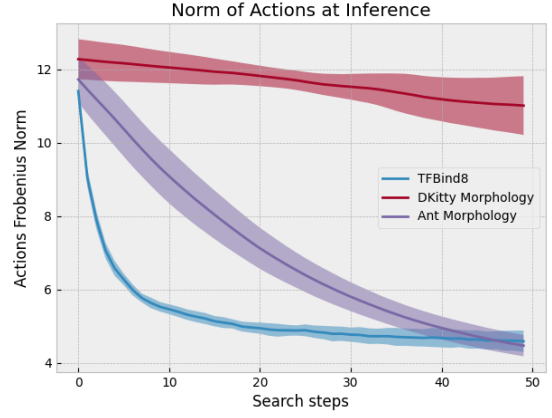


Figure 2: PGS Action Norms During Search Steps

we see that the performance drops as we increase the no. of steps (trajectory length). This decrease in performance can be attributed to the train-test mismatch in offline RL policy training since our policies are trained for trajectories of fixed length 50, which is aligned with the standard evaluation procedure in the offline optimization literature.

**PGS performance w/ varying dataset (number of trajectories) size for offline RL** Table 11 shows the ablation results for comparing PGS by training the policy with different dataset sizes (number of trajectories) using offline RL algorithm. Concretely, we ran CQL algorithm on three different sets of number of trajectories (1000, 10000, 20000) and evaluate the final performance. As evident from the table, the performance of PGS roughly increases as we increase the dataset size for offline RL training.

Table 7: Hyperparameters used for training the surrogate model and latent embedding for the OSEL procedure.

Hyperparameter	Discrete	Continuous
Number of epochs to train $\hat{f}_\theta$	50	50
Batch size to train $\hat{f}_\theta$	128	128
Adam learning rate to train $\hat{f}_\theta$	3e-4	3e-4
$T$ Number of gradient ascent steps in Equation 3	50	50
Discrete tasks latent vector dimension	32	-
Scale of $\alpha$ in Equation 4	$2.0\sqrt{d}$	$0.05\sqrt{d}$
KNN regressor k	10	10
KNN regressor k (breaking ties)	100	100
OSEL latent space dimension	8	32
OSEL embedding training window	8	8

Table 8: Selection of top  $p$  percentile and number of epochs for CQL training to train the policy for PGS for each task employed in the experiments.

Task	Ant	DKitty	Superconductor	TFBind8	UTR	GFP
Top p	20	40	40	30	10	40
CQL training epochs	300	300	100	150	50	50

Table 9: Selection of top  $p$  percentile and number of epochs for IQL training to train the policy for PGS for each task employed in the experiments.

Task	Ant	DKitty	Superconductor	TFBind8
Top p	40	20	40	40
IQL training epochs	100	50	100	300

Table 10: Table showing the ablation result of comparing different trajectory lengths (number of search steps at the test-time) guided by policy guided search during test-time search.

# Gradient Steps	50	60	70	80	90	100
AntMorphology	$0.95 \pm 0.02$	$0.91 \pm 0.02$	$0.91 \pm 0.02$	$0.91 \pm 0.03$	$0.92 \pm 0.04$	$0.89 \pm 0.04$
DKittyMorphology	$0.97 \pm 0.01$	$0.96 \pm 0.01$	$0.96 \pm 0.02$	$0.96 \pm 0.01$	$0.96 \pm 0.01$	$0.96 \pm 0.01$
Superconductor	$0.56 \pm 0.06$	$0.56 \pm 0.06$	$0.55 \pm 0.06$	$0.55 \pm 0.06$	$0.56 \pm 0.06$	$0.57 \pm 0.05$
TFBind8	$0.98 \pm 0.02$	$0.92 \pm 0.03$	$0.92 \pm 0.05$	$0.93 \pm 0.04$	$0.90 \pm 0.05$	$0.88 \pm 0.07$

Table 11: Table showing the ablation of different training dataset sizes (number of trajectories) for the offline RL algorithm in PGS.

Number of trajectories	1k	10k	20k
AntMorphology	$0.891 \pm 0.036$	$0.889 \pm 0.022$	$0.949 \pm 0.017$
DKittyMorphology	$0.943 \pm 0.010$	$0.964 \pm 0.014$	$0.966 \pm 0.013$
Superconductor	$0.535 \pm 0.061$	$0.521 \pm 0.059$	$0.563 \pm 0.058$
TFBind8	$0.829 \pm 0.022$	$0.900 \pm 0.080$	$0.981 \pm 0.015$