A NOTATIONS & THEORETICAL ANALYSIS

A.1 NOTATIONS

For ease of reference, we list the notations in Table 4.

Table 4: Important notations used in the paper

SYMBOL	MEANING
f	Black box function
$\mathcal{X}$	Support of f
$\mathbf{x}^*$	Optima (taken to be maxima for consistency)
$\mathcal{D}$	Offline dataset
N	Size of offline dataset
$\mathcal{D}_{\text{traj}}$	Trajectory dataset
num_trajs	Number of trajectories in $\mathcal{D}_{\text{traj}}$
$\mathcal{T}$	A trajectory
T	Length of a trajectory
Q	Query budget for black-box function
P	Prefix length
R_i	Regret Budget at timestep i
R_1	Initial Regret Budget
$\hat{R}$	Evaluation Regret Budget
g_θ	Autoregressive generative model with parameters θ
K, τ	SORT-SAMPLE hyperparameters
N_B	Number of bins
C	Context Length

A.2 THEORETICAL ANALYSIS

One crucial component in the SORT-SAMPLE is sorting the trajectories in the increasing order of the function values. Although our primary motivation for sorting is derived from the empirical observations from online black-box optimizers (Figure 1c), we note that for a certain class of functions that are non-trivial to solve from the perspective of optimization (maximization for our paper), lower (higher) function values occupy a larger (smaller) domain. Thus, intuitively we can relate lower function values with exploration and higher function values with exploitation. With this perspective, sorting can be seen as moving from a high-diversity region to a low-diversity region - a behavior typically seen in online black-box optimizers (Bijl et al., 2016; Garivier et al., 2016). In this section, we try to formally prove such properties for this constrained class of functions.

We consider the simplified case of differential 1D functions with certain assumptions for simplicity, and further extend this notion to a more general D-dimensional case. First, we define a notion of ϵ-high points.

Definition 1. *ϵ-high values. Let the range of f be denoted as $[y_{min}, y_{max}]$. Then, a function value y in this range is ϵ-high if $y \geq y_{min} + \epsilon(y_{max} - y_{min})$.*

Intuitively, the above definition implies that y is ϵ-high if it is in the top $1 - \epsilon$ fraction of the range of f. The following result characterizes the relative diversity of regions consisting of ϵ-high points for 1-D functions.

Proposition 1. *Let $f : [a, b] \to \mathbb{R}$, $a, b \in \mathbb{R}$, be a real-valued, continuous and differentiable function such that the $f(a)$ and $f(b)$ are not ϵ-high. Let $H \subseteq [a, b]$ be a Lebesgue-measurable set of x values for which $f(x)$ is ϵ-high, with $\epsilon > 0.5$. Let $H^c = [a, b] \setminus H$. Without loss of generality, let's assume that $f(a) \leq f(b)$. If the Lipchitz constant L of f is upper bounded by*

$$\frac{2(\epsilon(y_{max} - y_{min}) + y_{min} - f(a))}{b - a}, \tag{6}$$

then $|H| < |H^c|$, where $|\cdot|$ denotes volume of a set w.r.t. the Lebesgue Measure.

Proof of Proposition 1.

Note that if no point in the domain $[a, b]$ achieves ϵ-high function value, then the statement holds trivially true. So, we assume that there is atleast one point which has ϵ-high function value. Let x_1 and x_2 be the smallest and largest such points in the domain. Since the boundary points doesn't have ϵ-high values, $|H^c| \geq (x_1 - a) + (b - x_2)$ and $|H| \leq (x_2 - x_1)$. Thus, if we prove that $(x_1 - a) + (b - x_2) \geq (x_2 - x_1)$, then we are done. To prove this, we try to prove $(x_1 - a) \geq (x_2 - x_1)$.

Assume, on the contrary, that $(x_1 - a) < (x_2 - x_1)$. Rearranging, we get

$$\frac{2}{x_2 - a} < \frac{1}{x_1 - a} \tag{7}$$

Now, by the definition of Lipchitz constant, we have:

$$\begin{aligned} &\frac{f(x_1) - f(a)}{x_1 - a} \leq L \\ \Rightarrow &\frac{2(f(x_1) - f(a))}{x_2 - a} \overset{7}{\leq} L \\ \Rightarrow &\frac{2(\epsilon * (y_{max} - y_{min}) + y_{min} - f(a))}{b - a} \leq L \end{aligned} \tag{8}$$

Last inequality holds because $x_2 - a \leq b - a$. This inequality contradicts the bound 6 on L , completing our proof for Proposition 1. □

Now, we show an extension of this proposition for D-dimensional functions with hypercube domains.

Proposition 2. *Let $f : \mathcal{X} \to \mathbb{R}$ be a D-dimensional, real-valued, continuous, and differentiable function with hypercube domain $\mathcal{X} = [a_1, b_1] \times [a_2, b_2], \cdots, \times [a_D, b_D]$, such that none of the boundary points are ϵ-high, for some fixed ϵ. Here by boundary points, we mean the points on the surface of the domain hypercube. Let y_{max} and y_{min} be the maximum and the minimum values attained by f. Let $H \subseteq \mathcal{X}$ be a Lebesgue-measurable set of points for which $f(\mathbf{x})$ is ϵ-high. Let $H^c = \mathcal{X} \setminus H$. If the Lipchitz constant L of f is upper bounded by*

$$\frac{2(\epsilon * (y_{max} - y_{min}) + y_{min} - \max\limits_{x_2, \cdots, x_D} f(a_1, x_2, \cdots, x_D))}{b_1 - a_1}, \tag{9}$$

then $|H| < |H^c|$, where $|\cdot|$ denotes volume of a set w.r.t. the Lebesgue Measure.

Proof. We prove this proposition by induction on the number of dimensions D. Notice that for $D = 1$, the statement reduces to Proposition 1, which we have already proved. Next, we assume that the statement holds for $(D - 1)$-dimensional functions and prove it for D dimensions, with $D > 1$.

Let's define $\mathcal{H}_{D,\epsilon} : \mathcal{F}_D \to \mathcal{B}_D$ to be a functional that maps any D-dimensional function, say f, to a Lebesgue-measurable subset of $\mathbb{R}^D$ that corresponds to the set of points where $f(\mathbf{x})$ is ϵ-high. , Here, $\mathcal{F}_D$ and $\mathcal{B}_D$ denote the set of all D-dimensional functions and the set of all Lebesgue-measurable subsets of $\mathbb{R}^D$ respectively.

We similarly define the mapping $\mathcal{H}^c_{D,\epsilon}$ to be a functional mapping a function f to the complement of its ϵ-high region. Thus, $H = \mathcal{H}_{D,\epsilon}(f)$ and $H^c = \mathcal{H}^c_{D,\epsilon}(f)$. Now, by definition,

$$|\mathcal{H}_{D,\epsilon}(f(\bullet, \cdots, \bullet)| = \int_{x_D} |\mathcal{H}_{D-1,\epsilon}(f(\bullet, \cdots, \bullet, x_D))| \, dx_D \tag{10}$$

And similarly,

$$|\mathcal{H}^c_{D,\epsilon}(f(\bullet, \cdots, \bullet)| = \int_{x_D} |\mathcal{H}^c_{D-1,\epsilon}(f(\bullet, \cdots, \bullet, x_D))| \, dx_D \tag{11}$$

Consequently, to prove $|\mathcal{H}_{D,\epsilon}(f(\bullet, \cdots, \bullet)| \leq |\mathcal{H}^c_{D,\epsilon}(f(\bullet, \cdots, \bullet)|$, we prove $|\mathcal{H}_{D-1,\epsilon}(f(\bullet, \cdots, \bullet, x_D))| \leq |\mathcal{H}^c_{D-1,\epsilon}(f(\bullet, \cdots, \bullet, x_D))|$ for every $x_D \in [a_D, b_D]$.

To do this, we first fix the D^{th} dimension to be c. In other words, we are considering the $(D-1)$-dimensional slice of $f(x_1, \cdots, x_D)$ with $x_D = c$. Let g be such a slice with $g(x_1, \cdots, x_{D-1}) = f(x_1, \cdots, x_{D-1}, c)$. First we need ϵ^g for which ϵ^g-high value for g is ϵ-high for f:

$$\epsilon^g(y^g_{max} - y^g_{min}) + y^g_{min} = \epsilon(y_{max} - y_{min}) + y_{min} \tag{12}$$

where y^g_{max} and y^g_{min} are the minimum and maximum values respectively achieved by g. By this choice of ϵ^g, we are ensuring that a point $(x_1, \cdots, x_{D-1}, c)$ is ϵ-high w.r.t f if and only if $(x_1, \cdots, x_{D-1})$ is ϵ^g-high w.r.t g. In other words,

$$\begin{aligned} \mathcal{H}_{D-1,\epsilon^g}(g) &= \mathcal{H}_{D-1,\epsilon}(f(\bullet, \cdots, \bullet, c)) \\ \mathcal{H}^c_{D-1,\epsilon^g}(g) &= \mathcal{H}^c_{D-1,\epsilon}(f(\bullet, \cdots, \bullet, c)) \end{aligned} \tag{13}$$

Let the Lipchitz constant of g be L^g. First we show that $L_g \leq L$. By definition of Lipchitz constant, for $\mathbf{x} = (x_1, \cdots x_{D-1}), \mathbf{z} = (z_1, \cdots, z_{D-1})$ in the domain of g,

$$\begin{aligned} L_g &= \sup_{\mathbf{x} \neq \mathbf{z}} \frac{|g(x_1, \cdots, x_{D-1}) - g(z_1, \cdots, z_{D-1})|}{\sqrt{\sum_{i=1}^{D-1}(x_i - z_i)^2}} \\ &= \sup_{\mathbf{x} \neq \mathbf{z}} \frac{|f(x_1, \cdots, x_{D-1}, c) - f(z_1, \cdots, z_{D-1}, c)|}{\sqrt{\sum_{i=1}^{D-1}(x_i - z_i)^2 + (c-c)^2}} \\ &\leq L \end{aligned} \tag{14}$$

Where last inequality is by definition of L w.r.t f. Combining this with our bound on L in 9, we get

$$\begin{aligned} L_g &\leq \frac{2(\epsilon * (y_{max} - y_{min}) + y_{min} - \max_{x_2, \cdots, x_D} f(a_1, x_2, \cdots, x_D))}{b_1 - a_1} \\ &\leq \frac{2(\epsilon * (y_{max} - y_{min}) + y_{min} - \max_{x_2, \cdots, x_{D-1}} f(a_1, x_2, \cdots, c))}{b_1 - a_1} \text{ (fixing } D^{th} \text{ dimension)} \\ &\overset{12}{\leq} \frac{2(\epsilon^g * (y^g_{max} - y^g_{min}) + y^g_{min} - \max_{x_2, \cdots, x_{D-1}} g(a_1, x_2, \cdots, x_{D-1}))}{b_1 - a_1} \end{aligned} \tag{15}$$

Thus, the Lipchitz bound assumption is followed by g with $\epsilon = \epsilon^g$. Also, the boundaries are not ϵ^g-high w.r.t g because of the choice of ϵ^g. This implies, by inductive assumption, that $|\mathcal{H}_{D-1,\epsilon^g}(g)| \leq |\mathcal{H}^c_{D-1,\epsilon^g}(g)|$. This, combined with equality 13 proves that that $|\mathcal{H}_{D-1,\epsilon}(f(\bullet, \cdots, \bullet, c))| \leq |\mathcal{H}^c_{D-1,\epsilon}(f(\bullet, \cdots, \bullet, c))|$. Since this is true for all $c \in [a_D, b_D]$, by equations 10 and 11, our proof for $|\mathcal{H}_{D,\epsilon}(f(\bullet, \cdots, \bullet)| \leq |\mathcal{H}^c_{D,\epsilon}(f(\bullet, \cdots, \bullet)|$ is complete.

□

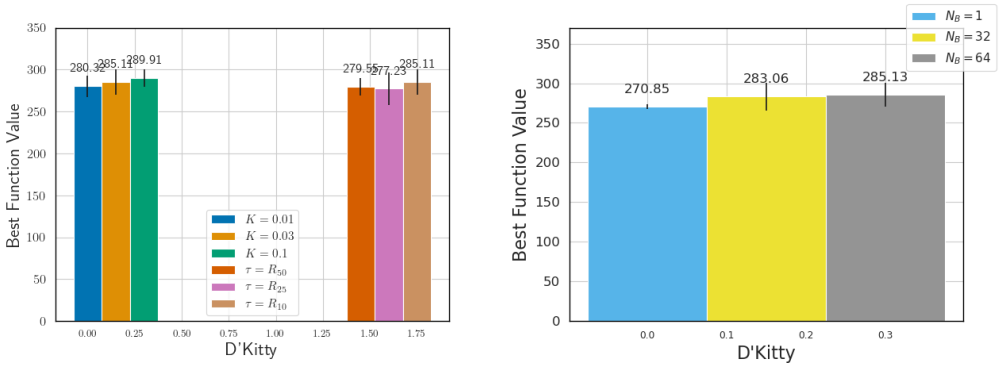
B Experimental Details

B.1 SORT-SAMPLE

In SORT-SAMPLE, the score of each bin is calculated according to the formula

$$s_i = \frac{|B_i|}{|B_i| + K} \exp\left(\frac{-|\hat{y} - y_{b_i}|}{\tau}\right)$$

The two variables K and τ here act as smoothing parameters. K controls the relative priority given to the larger bins (bins with more points). Higher value of K assigns higher relative weight to these large bins compared to smaller bins, whereas a low value of K the weight assigned to large and small bins would be similar. In the extreme case where $K = 0$, the weight due to $\frac{|B_i|}{|B_i|+K}$ will always be 1, regardless of bin size. For very large value of K, the weight will be approximately linearly proportional to the bin size $|B_i|$. The later case is not desirable because if there is a bin



(a) Varying K and τ

(b) Varying N_B

Figure 6: We plot the best achieved function value for different combinations of K, τ and N_B. As expected, we don't see much sensitivity to the choice of K and τ. For N_B, as expected, we a significant decrease for $N_B = 1$, but $N_B = 32$ is comparable to $N_B = 64$

which has very large number of points (which might be a low quality bin), then most of the total weight will be given to that bin because of the linear proportionality.

Temperature τ controls how harshly the bad bins are penalized. Lower the τ value, lower the relative score of the low quality bins (bins with high regret) and vice versa.

In our experiments, we don't tune the values of K, τ, and number of bins N_B. In all the tasks, we use $K = 0.03 \times N$ and $\tau = R_{10}$, where R_{10} is the 10^{th} percentile regret value in $\mathcal{D}$. For Branin task, we use , $N_B = 32$ and for all the Design-Bench tasks, we use $N_B = 64$. Empirically, as we show in Figure 6a, we didn't observe much effect on K in the range $[0.01, 0.1]$, and for τ from the 50^{th} to the 10^{th} percentiles of R. Figure 6b shows variation with N_B, keeping all other parameters fixed. As expected, $N_B = 1$ doesn't perform well, as having just one bin is equivalent to having no re-weighting. Beyond 32, we don't see much variation with the value of N_B.

B.2 Model architecture & implementation

Architecture We use a GPT (Radford et al., 2019) like architecture, where each timestep refers to two tokens R_t and $\mathbf{x}_t$. Similar to Chen et al. (2021), we add a new learned timestep embedding (in addition to the positional embedding already present in transformers). Each token R_t and $\mathbf{x}_t$ that goes as input to our model is first projected into a 128 dimensional embedding space using a linear embedding layer. To this embedding, we also add the positional and timestep embeddings. This is passed through a causally masked transformer. The prediction head for R_t predicts $\hat{\mathbf{x}}_t$, which is then used to compute the loss. The output of the prediction head for $\mathbf{x}_t$ is discarded. At each timestep, we feed in the last C timesteps to the model, where C here refers to the context length. For continuous tasks, the prediction head for R_t outputs a d-dimensional prediction $\hat{\mathbf{x}}_t$. For discrete tasks, the prediction head outputs a $V \times d$-dimensional prediction, where V refers to the number of classes in the discrete task. Thus, each dimension in $\mathcal{X}$ corresponds to a V-dimensional logits vector.

Code Our code (available at the anonymized link here) is built upon the code from minGPT [3] and Chen et al. (2021) [4]. All code we use is under the MIT licence.

Training The parameter details for all the tasks are summarized in the Table 5. Note that almost all of the parameters are same across all the Design-Bench tasks. Number of layers is higher for continuous tasks, as they are of higher dimensionalities. For all the tasks, we use a batch size of 128 and a fixed learning rate of 10^{-4} for 75 epochs. All training is done using 10 Intel(R) Xeon(R) CPU cores (E5-2698 v4443 @ 2.20GHz) and one NVIDIA Tesla V100 SXM2 GPU.

[3] https://github.com/karpathy/minGPT

[4] https://github.com/kzl/decision-transformer

B.3 EVALUATION

For all the Design-Bench tasks (except NAS), we use a query budget $Q = 256$. For NAS, we use a query budget of $Q = 128$ due to compute restrictions. Since we use a trajectory length of 128 and a prefix length of 64, this means that we can roll-out four different trajectories. There are two variable parameters during the evaluation: Evaluation RB ($\hat{R}$) and the prefix sub-sequence. We empirically observed that $\hat{R}$ has more impact on the variability of rolled-out points compared to prefix sub-sequence. Hence, we roll-out trajectory for 4 different low $\hat{R}$ values $(0.0, 0.01, 0.05, 0.1)$. These values are kept fixed across all the tasks and are not tuned. They are chosen to probe the interval $[0.0, 0.1]$, while giving slightly more importance to the low values by choosing 0.01. Evaluation strategies with lower query budget Q available are discussed in the section C.2

Table 5: Important parameters for all the tasks

TASK	**Type**	**HEADS**	**LAYERS**	T	P	C	N_B	num_trajs
Branin (toy)	Continuous	4	8	64	32	32	32	400
TFBind8	Discrete	8	8	128	64	64	64	800
TFBind10	Discrete	8	8	128	64	64	64	800
ChEMBL	Discrete	8	8	128	64	64	64	800
NAS	Discrete	8	8	128	64	64	64	800
D'Kitty	Continuous	8	32	128	64	64	64	800
Ant	Continuous	8	32	128	64	64	64	800
Superconductor	Continuous	8	32	128	64	64	64	800

B.4 FIXED VS. UPDATED RB DURING EVALUATION

During the evaluation of the prediction sequence, we proposed to keep the Regret Budget (RB) fixed. One alternative strategy is to sequentially update RB after every iteration, i.e. predict $\mathbf{x}_t$ from R_t, and compute $R_{t+1} = R_t - (f(\mathbf{x}^*) - f(\mathbf{x}_t))$. However, there are two issues with such an update rule:

1. Updating RB adds a sequential dependency on our model during evaluation, as we must query $f(\mathbf{x}_t)$ to compute R_{t+1}. Thus, generating the Q candidate points is not purely offline.
2. While updating the regret budget R_t, it is possible that at some timestep t, R_t becomes negative. Since the model has never seen negative RB values during training, this is undesirable.

Hence, we do not update RB during evaluation, and instead provide a fixed $\hat{R}$ value at every timestep after the prefix length. This way, point proposal is not dependent on sequential evaluations of f, making it much faster as the evaluations on f can then be parallelized. Furthermore, by not updating $\hat{R}$ we sidestep the issue of negative RBs. Empirically, as we see in Figure 7, there is not much difference across different strategies, which justifies our choice of not updating RB, allowing our method to be purely offline.

B.5 BASELINES

For the gradient ascent baseline of Branin task, we train a 2 layer neural network (with hidden layer of size 128) as a forward model for 75 epochs with a fixed learning rate of 10^{-4}. For gradient ascent on the learnt model during evaluation, we report results with a step size of 0.1 for 64 steps. We average over 5 seeds, and for each seed we perform two random restarts.

For the baselines in the Design-Bench tasks, we run the baseline code [5] provided in Trabucco et al. (2022) and report results with the default parameters for a query budget of 256.

[5]We were not able to reproduce the results of (Fu & Levine, 2021) and (Yu et al., 2021) on the latest version of Design-Bench.

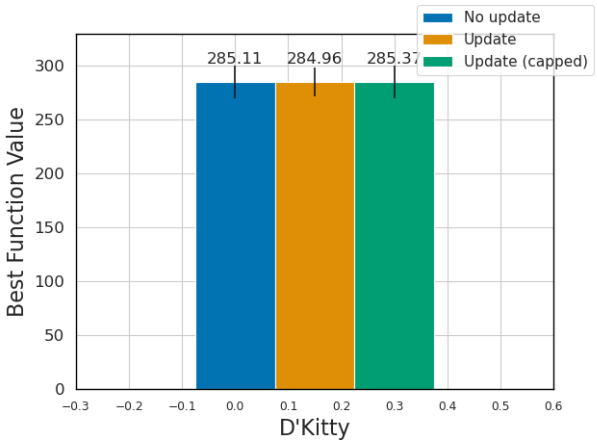


Figure 7: Comparison between 3 strategies: (1) do not update the RB at all (blue), (2) update the RB but do not handle the case when R_t becomes negative (orange) and (3) don't make the update if the update is going to make the RB negative and update otherwise (green). In all three cases, we see similar performance.

B.6 Design-Bench Tasks

For the Design-Bench tasks, we pre-process the offline dataset to normalize the function values before constructing the trajectory dataset $\mathcal{D}_{\text{traj}}$. Normalized y_{norm} values are computed as $y_{norm} = \frac{y - y_{min}}{y_{max} - y_{min}}$, where y_{min} and y_{max} are minimum and maximum function values of a much larger hidden dataset. Note that we only use the knowledge of y_{min} and y_{max} from this hidden dataset, and not the corresponding $\mathbf{x}$ values. With this normalization procedure, we use 1.0 as an estimate of $f(\mathbf{x}^*)$ while constructing trajectories in $\mathcal{D}_{\text{traj}}$.The results we report in Table 2 are also normalized using the same procedure, similar to prior works (Trabucco et al., 2022; 2021). We also report unnormalized results in Table 7.

Table 6: Task details

TASK	SIZE	DIMENSIONS	TASK MAX
TFBind8	32898	8	1.0
TFBind10	10000	10	2.128
ChEMBL	1093	31	443000.0
NAS	1771	64	69.63
D'Kitty	10004	56	340.0
Ant	10004	60	590.0
Superconductor	17014	86	185.0

B.7 Hopper Task

We didn't include the Hopper task in our results in Table 2 because of the inconsistency between the offline dataset values and the oracle outputs. Hopper data consist of 3200 points, each with 5126 dimensions. The lowest and highest function values are 87.93 and 1361.61. Figure 8 shows the distribution of the normalized function values. This distribution is extremely skewed towards low function values. Only 6 points out of 3200 have a normalized function value greater than 0.5.

We noticed that the oracle of Hopper is highly inaccurate for points with higher function values. Figure 9 shows the function values in the dataset vs. the oracle output for the top 10 best points in the data, clearly showing the inconsistency between the two. In fact, the oracle minima and maxima for the dataset are just 56.26 and 786.79, respectively, far from the actual dataset values. Due to such discrepancies, we have decided not to include the Hopper task in our analysis.

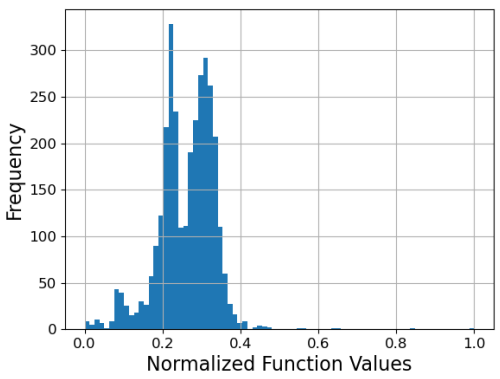


Figure 8: Histogram of normalized function values in the Hopper dataset. The distribution is highly skewed towards low function values.

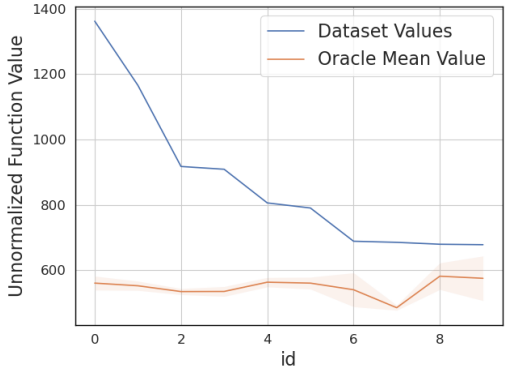


Figure 9: Dataset values vs Oracle values for top 10 points. Oracle being noisy, we show mean and standard deviation over 20 runs.

C Additional Experimental Results

C.1 Ablations on SORT-SAMPLE Strategy

SORT-SAMPLE algorithm has two main components: Sampling after re-weighting and sorting. Our sorting heuristic is primarily motivated by typical runs of online optimizers. To show this, we run an online GP to optimize the three synthetic functions, namely the negative Branin, negative Goldstein-Price and negative Six hump camel functions and plot the function values for the proposed points. Figure 10 shows sample trajectories of the function values of the proposed maxima after each function evaluation. We can see, on average, the function values tend to increase over time as the number of queries increases. Such behavior has also been reported for other black-box functions and setups, see e.g. (Bijl et al., 2016).

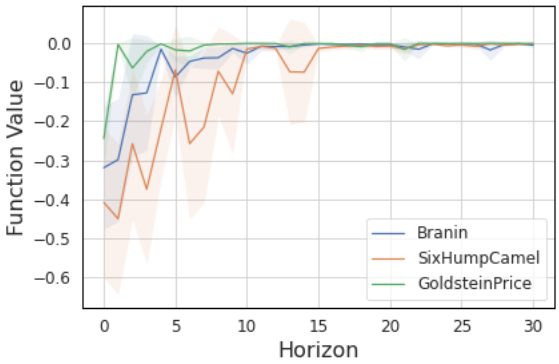


Figure 10: Mean and standard deviations of 10 trajectories unrolled by a simple GP-based BayesOpt algorithm on the 3 synthetic functions.

However, in this section, we do perform ablations to see the effects of these components. To this end, we construct trajectories using 4 strategies:

1. Random: Uniformly randomly sample a trajectory from the offline dataset.
2. Random + Sorting: Uniformly randomly sample a trajectory from the offline dataset and sort it in ascending order of the function values.
3. Re-weighting + Partial Sorting: Perform re-weighting, uniformly randomly sample n_i number of points from each bin, and concatenate them from lowest quality bin to the high-

est quality bin. This way, the trajectory will be partially sorted, i.e. the order of the bins themselves is sorted, but the points sampled from a bin will be randomly ordered. In this case, the trajectories are not entirely monotonic w.r.t. the function values. Intuitively, this intermingles exploration and exploitation phases within and across bins respectively.

4. Re-weighting + Sorting (default in BONET): Sort the trajectory obtained in strategy 3. This is the default setting we use in our experiments.

Figure 11 contains the results obtained by each of the four strategies. Note that while going from strategy 1 to 2, we keep the points sampled in a trajectory the same, so the only difference between them is sorting. Figure 11 shows that strategy 1 clearly outperforms strategy 2. This means that sorting has a significant impact on the results. Next, note that strategy 2 and 4 differ only in their sampling strategy, and strategy 4 outperforms strategy 2, which shows the effectiveness of re-weighting. This experiment justifies our choice for both re-weighting and sorting.

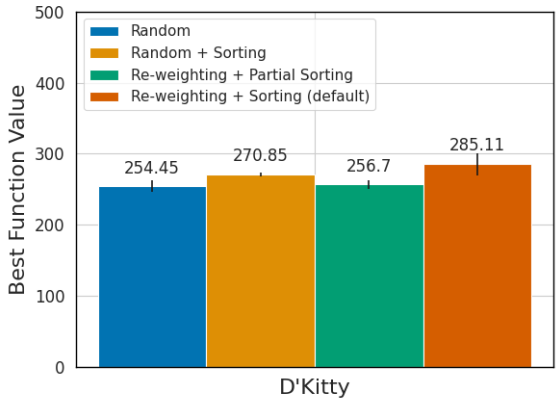


Figure 11: Results with various trajectory construction strategies for D'Kitty task, averaged over 5 runs. Comparing blue and orange bars, it is evident that sorting is improving the results. Similarly, comparison of orange bar with red bar shows that re-weighting further improves the results.

C.2 Analysis on query budget Q

So far, we have been discussing the results with query budget $Q = 256$. Here, we describe the evaluation strategy we use when lower query budget is available. Our strategy will be to give higher preference to lower $\hat{R}$ values when lower budget is available. For example, when $Q = 192$, we only roll-out and evaluate for $\hat{R} \in \{0.0, 0.01, 0.05\}$. For $192 < Q \leq 256$, we will roll-out for $\hat{R} \in \{0.0, 0.01, 0.05, 0.1\}$, evaluate the entire predicted sub-sequences of lengths 64 for $\{0.0, 0.01, 0.05\}$, and evaluate the first $Q - 192$ points in the predicted sub-sequences for $\hat{R} = 0.1$. In the Figure 12, we present the results for different query budgets for our method compared to important baselines, for the D'Kitty task. We outperform the baselines for almost all the query budget values.

C.3 Additional Ablations

Here we present ablations similar to Section 3 on the D'Kitty task, and observe similar trends to what we see in the Branin ablations.

C.4 Effect of Prefix Sequences

During the evaluation, the unrolled trajectory depends on two factors affecting the unrolled output: Evaluation Regret Budget $\hat{R}$ and the prefix sequence. Empirically, we found that $\hat{R}$ has a larger impact on the unrolled trajectory than the prefix sequence. To show this, we first evaluate the Branin task for 10 different randomly sampled prefix sequences for a fixed $\hat{R}$ and then do the same with

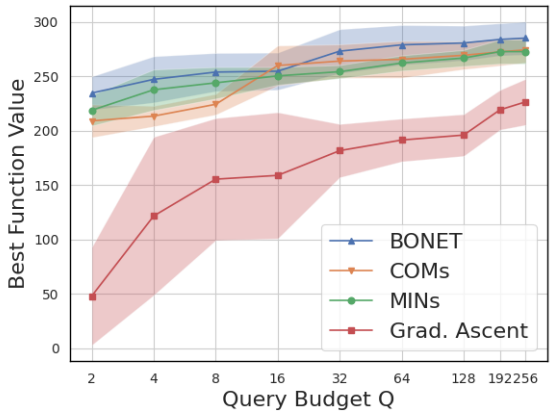
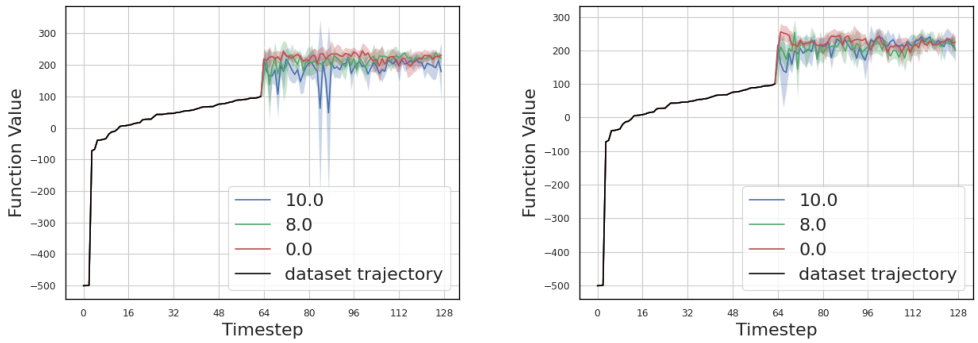


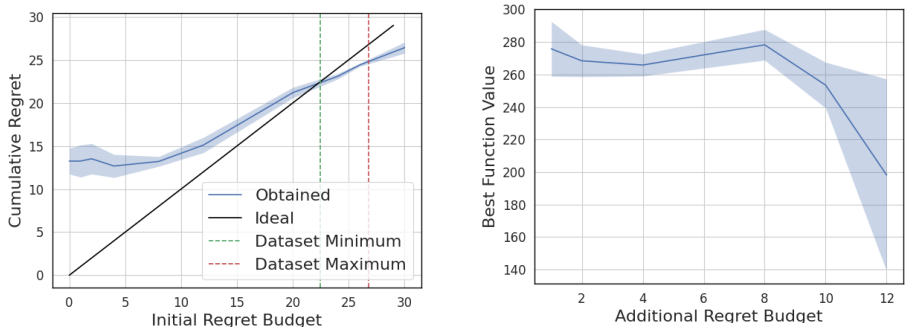
Figure 12: Results for various query budget values Q for D'Kitty task, averaged over 5 runs. We match or outperform other baselines on almost all the values of Q.



(a) $P = 64$, without updating RB

(b) $P = 64$, with updating RB

Figure 13: Figures 13a and 13b show plots of the trajectories generated on DKitty for different values of evaluation RB (0, 8 and 10). In Figure 13a we show results without updating RB, and in Figure 13b we show results with updating. All the trajectories are averaged over 5 runs.



(a) Initial RB vs Cumulative Regret

(b) Effect of $\hat{R}$

Figure 14: Ablations on D'Kitty, averaged over 5 runs.

10 different $\hat{R}$ values sampled from the range $(0.0, 0.5)$ for the same prefix sequence. Figure 15 shows the standard deviation of the minimum regret of the 10 different unrolled trajectories for 3

Table 7: Unnormalized **100th percentile** results

BASELINE	TFBIND8	TFBIND10	SUPERCONDUCTOR	ANT	D'KITTY	CHEMBL	NAS
$\mathcal{D}$ (best)	0.439	0.00532	74.0	165.326	199.231	443000.000	63.79
CbAS	0.958 ± 0.018	0.761 ± 0.067	83.178 ± 15.372	468.711 ± 14.593	213.917 ± 19.863	389000.000 ± 500.000	66.355 ± 0.79
GP-qEI	0.824 ± 0.086	0.675 ± 0.043	$\mathbf{92.686 \pm 3.944}$	480.049 ± 0.000	213.816 ± 0.000	387950.000 ± 0.000	$\mathbf{69.722 \pm 0.59}$
CMA-ES	0.933 ± 0.035	$\mathbf{0.848 \pm 0.136}$	90.821 ± 0.661	$\mathbf{1016.409 \pm 906.407}$	4.700 ± 2.230	388400.000 ± 400.000	$\mathbf{69.475 \pm 0.79}$
Gradient Ascent	$\mathbf{0.981 \pm 0.010}$	0.770 ± 0.154	$\mathbf{93.252 \pm 0.886}$	-54.955 ± 33.482	226.491 ± 21.120	390050.000 ± 2000.000	63.772 ± 0.000
REINFORCE	0.959 ± 0.013	0.692 ± 0.113	89.027 ± 3.093	-131.907 ± 41.003	-301.866 ± 246.284	388400.000 ± 2100.000	39.724 ± 0.000
MINs	0.938 ± 0.047	0.770 ± 0.177	89.469 ± 3.227	533.636 ± 17.938	272.675 ± 11.069	$\mathbf{390950.000 \pm 200.000}$	66.076 ± 0.46
COMs	0.964 ± 0.020	0.750 ± 0.078	78.178 ± 6.179	540.603 ± 20.205	$\mathbf{277.888 \pm 7.799}$	390200.000 ± 500.000	64.041 ± 1.390
BONET	$\mathbf{0.975 \pm 0.004}$	$\mathbf{0.855 \pm 0.139}$	80.84 ± 4.087	$\mathbf{567.042 \pm 11.653}$	$\mathbf{285.110 \pm 15.130}$	$\mathbf{391000.000 \pm 1900.000}$	66.779 ± 0.16

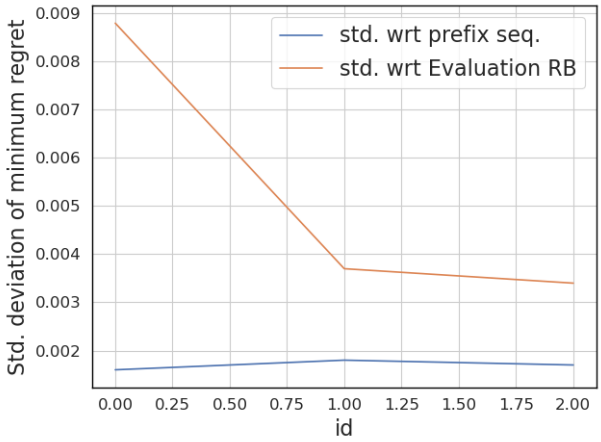


Figure 15: Standard deviation of the minimum regret of the unrolled trajectories with 1) 10 different prefix sequences for a fixed $\hat{R}$ and 2) 10 different $\hat{R}$ for a fixed prefix sequence.

fixed $\hat{R}$ and prefix sequences, respectively. The standard deviation for the variation in prefix length is consistently lower than that for the variation in $\hat{R}$, explaining our choice of spending query budget on different $\hat{R}$ rather than different prefix sequences.

C.5 Effect of estimating $y_{\max}$

A key assumption of our method is the knowledge of $y_{\max}$. Though in many problems this is not an issue, there are many other problems where the value of $y_{\max}$ may not be known. A simple solution could be to just estimate $y_{\max}$. In Figure 16 we evaluate BONET on D'Kitty multiple varying values of $y_{\max}$ starting from just beyond the dataset maxima. We find that the value of y_{max} initially affects performance alot, but beyond a point, it plateaus..

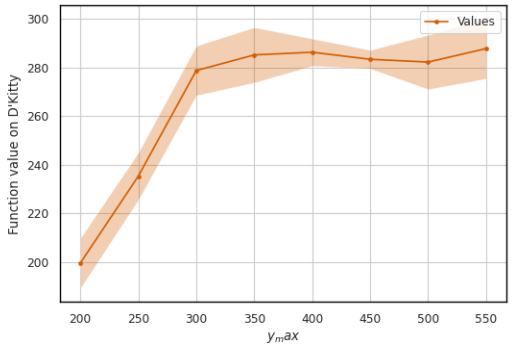


Figure 16: Best points for different values of $y_{\max}$ on D'Kitty.

Table 8: Results for when a random $x\%$ subsection of the offline dataset was withheld during training from BONET

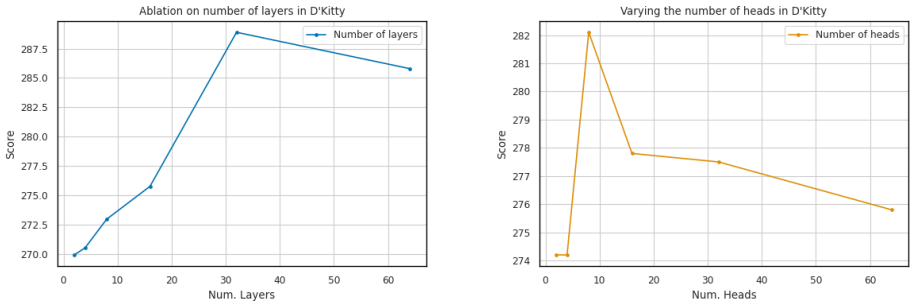
	10%	**50%**	**90%**	**99%**
$\mathcal{D}$ (best)	74.21	74.20	73.99	65.71
BONET	286.60 ± 1.47	284.74 ± 23.68	274.11 ± 7.57	241.17 ± 18.07

Table 9: Results for when the top $x\%$ of the offline dataset was withheld during training from BONET

	10%	**50%**	**90%**	**99%**
$\mathcal{D}$ (best)	61.14	-40.40	-545.36	-548.89
BONET	267.68 ± 2.28	261.04 ± 28.09	211.56 ± 16.74	193.27 ± 5.51

C.6 Ablation on model parameters

In this experiment we study the effect of changing the number parameters in BONET. We do this by altering the number of layers and heads in BONET on D'Kitty. We find that increasing the number of parameters helps up to a point, beyond which the model over-fits. It is important to note that we present this study only to understand the impact of model size on our performance. We don't actually tune over these parameters in our experiments. They are kept fixed across all the discrete and continuous tasks (refer to Table 5).



(a) Varying the number of layers in BONET. Heads are fixed at 8

(b) Varying the number of heads in BONET. Layers are fixed at 16

Figure 17: We show the performance of various models with differing number of layers and heads on D'Kitty to see their effect on BONET. We find that increasing the number of parameters helps upto a point, beyond which it overfits.

C.7 Ablations on Dataset Size

To test the limits of BONET we run experiments where we withhold offline training data from BONET and evaluate the performance. We have two settings, one where we withhold an $x\%$ size random subsection of data in Table 8, and another where we withhold the top x percentile of data during training and evaluation in Table 9. We see that with just reducing the number of points, we don't see as sharp of a drop off in performance as compared to when we withhold the good points in the dataset. This leads us to believe that the dominating effect is not the size of the dataset exactly, but the quality of points in the dataset. Further, note that even here the points proposed are significantly larger than the maximum point in the dataset, which rules out the possibility of memorization for BONET.

C.8 NOISE ABLATION

We run an experiment where we add progressive larger amounts of noise to the y values in our offline dataset while training our model, to test how robust BONET is to noisy data. We report the results in Table 10 for D'Kitty. We find that, as expected, increasing noise reduces performance, and BONET is in fact reasonably robust to noise, and the largest drop-off occurs when the magnitude of noise is equal to the magnitude of values.

Table 10: Ablation on adding various magnitudes of noise to training data

NOISE SCALE	SCORE
0% of max	285.110
2% of max	279.746
20% of max	255.925
100% of max	137.485

C.9 RANDOM BASELINE

One might argue that BONET just memorizes the best points in the offline dataset and outputs random points close to those best points during evaluation. To rule out this possibility, we perform a simple experiment for the D'Kitty task. We choose a small hypercube domain around the optimal point in the offline dataset and uniformly randomly sample 256 points in that domain. In Table 11, we show the results for different widths of this hypercube. 0 width means only the best point in the dataset.

For smaller hypercubes around the best points in the offline dataset, we see that the best point found by 256 random searches is roughly 225, which is significantly lower than what BONET finds (291.08). For larger hypercubes, the points are highly diverse. These observations suggest that this optimization problem cannot be solved by just randomly outputting points around the best point in the dataset. If we look at the 256 points output by BONET, they are consistently good (mean is 220), with comparatively very low variance. This suggests that BONET is not simply outputting random points around the best points in the dataset.

D ADDITIONAL ANALYSIS

D.1 VIZUALIZATION OF PREDICTED POINTS

Here we try to visualize the predicted points of BONET compared to the points in the offline data to study the nature of the points proposed by the model. As shown in Figure 18, BONET generalizes well on the unseen maxima regions of the function and produces low regret points.

D.2 ACTIVE GP EXPERIMENT

We also run a experiment to compare BONET with an active BBO method. Namely, we compare BONET with active BayesOpt, using the same GP prior and acquisition function (quasi-Expected Improvement) as mentioned in Section 3. The difference between the active method and the offline method we compare with in Table 2 is that while the active method directly optimizes the ground truth function, the offline method first trains a surrogate neural network on the data, and then performs bayesian optimization on the surrogate instead of the ground truth function. This is done to make the BayesOpt baseline fully offline, and is the same procedure followed by Trabucco et al. (2022; 2021). Note that this would result in an unfair comparison since the method is both online and queries the oracle function resulting in it using more queries than our budget. As shown in Table 12, We find that using oracle actually doesn't necessarily improve performance across all tasks, and there are other tasks where the performance doesn't change at all. And our model does outperform even the oracle GP-qEI method on several tasks.

Table 11: Results of using a simple sampling strategy randomly from a small hypercube centered around the optima. We find that BONET considerably beats this baseline, indicating that generalization occurring with BONET is not fortuitous.

Width of hypercube	**Max. function Value**	**Mean function Value**	**Std. Deviation**
0.0	199.23	199.23	0.0
0.005	212.66	190.68	9.28
0.01	222.44	182.13	12.21
0.05	226.10	-169.62	331.00
0.1	209.00	-368.71	261.37
BONET	291.08	221.00	24.43

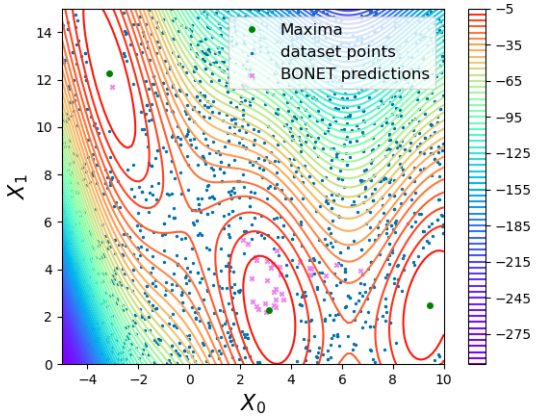


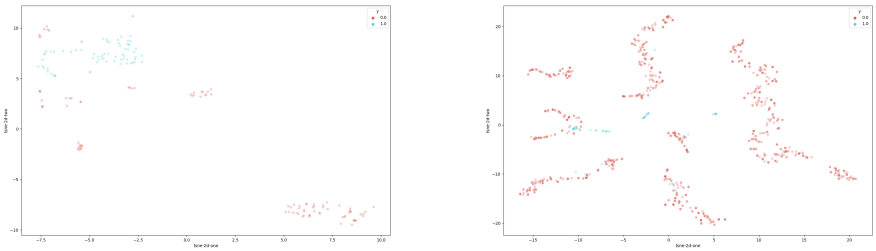
Figure 18: Vizualization of 32 points unrolled by a sample evaluation trajectory of BONET compared to 2000 points randomly sampled from the offline dataset. Three maxima regions don't contain any dataset points because of the removal of the top 10%-ile of uniformly sampled points, as described in the section 3.1. However, almost all the unrolled points fall into a maxima region, clearly showing the generalization capability of BONET.

Table 12: Comparison with GP-qEI on oracle function

	TFBIND8	**TFBIND10**	**SUPERCONDUCTOR**	**ANT**	**D'KITTY**
GP-qEI (active)	0.945 ± 0.018	0.922 ± 0.231	94.587 ± 2.137	480.049 ± 0.000	213.816 ± 0.000
GP-qEI	0.824 ± 0.086	0.675 ± 0.043	92.686 ± 3.944	480.049 ± 0.000	213.816 ± 0.000
BONET	0.975 ± 0.004	0.855 ± 0.139	80.84 ± 4.087	567.042 ± 11.653	285.110 ± 15.130

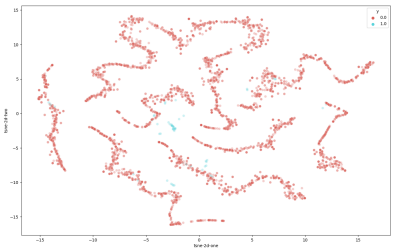
D.3 T-SNE PLOTS ON D'KITTY

We show t-SNE plots on DKitty for the datasets of differing sizes, with the removal of randomly sampled $x\%$ of the data (setting one described in the previous section). The blue points represent points proposed by our model, and the red points represent points in the dataset. In general, blue points do not overlap the red points, indicating that the points proposed by BONET are from a different region.

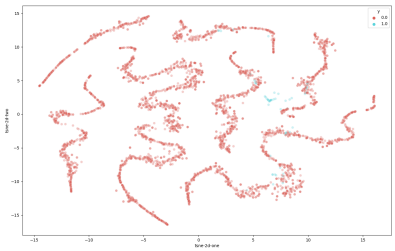


(a) 99%

(b) 90%



(c) 50%



(d) 10%

Figure 19: We show tSNE plots on DKitty for the datasets of differing sizes with the removal of randomly sampled $x\%$ of the data. The blue points represent points proposed by our model, and the red points represent points in the dataset. We find that in general, blue points do not overlap the red points, indicating that the points proposed by BONET are from a different region.