Bootstrapped Training of Score-Conditioned Generator for Offline Design of Biological Sequences

Minsu Kim [1] **Federico Berto** [1] **Sungsoo Ahn** [2] **Jinkyoo Park** [1]

Abstract

We study the problem of optimizing biological sequences, e.g., proteins, DNA, and RNA, to maximize a black-box score function that is only evaluated in an offline dataset. We propose a novel solution, bootstrapped training of score-conditioned generator (BOOTGEN) algorithm. Our algorithm repeats a two-stage process. In the first stage, our algorithm trains the biological sequence generator with rank-based weights to enhance the accuracy of sequence generation based on high scores. The subsequent stage involves bootstrapping, which augments the training dataset with self-generated data labeled by a proxy score function. Our key idea is to align the score-based generation with a proxy score function, which distills the knowledge of the proxy score function to the generator. After training, we aggregate samples from multiple bootstrapped generators and proxies to produce a diverse design. Extensive experiments show that our method outperforms competitive baselines on biological sequential design tasks.

1. Introduction

The automatic design of biological sequences, e.g., DNA, RNA, and proteins, with a specific property, e.g., high binding affinity, is a vital task within the field of biotechnology (Barrera et al., 2016; Zimmer, 2002; Sample et al., 2019; Lorenz et al., 2011). To solve this problem, researchers have developed algorithms to optimize a biological sequence to maximize a score function (Ren et al., 2022; Brookes et al., 2019; Brookes & Listgarten, 2018; Angermueller et al., 2019; Jain et al., 2022). Here, the main challenge is the expensive evaluation of the score function that requires experiments in a laboratory setting or clinical trials.

To resolve this issue, recent works have investigated offline model-based optimization (Kumar & Levine, 2020; Fu & Levine, 2021; Trabucco et al., 2021; Yu et al., 2021; Chen et al., 2022; Trabucco et al., 2022, MBO). Given an offline dataset of biological sequences paired with scores, offline MBO algorithms train a proxy for the score function, e.g., a deep neural network (DNN), and maximize the proxy function without querying the true score function. Therefore, such offline MBO algorithms bypass the expense of iteratively querying the true score function whenever a new solution is proposed. However, even optimizing such a proxy function is challenging due to the vast search space over the biological sequences.

Related works. On one hand, several works (Fu & Levine, 2021; Trabucco et al., 2021; Yu et al., 2021; Chen et al., 2022) considered applying gradient-based maximization of the proxy function. However, when the proxy function is parameterized using a DNN, these methods often generate solutions where the true score is low despite the high proxy score. This is due to the fragility of DNNs against *adversarial* optimization of inputs (Yu et al., 2021; Trabucco et al., 2021; Fu & Levine, 2021). Furthermore, the gradient-based methods additionally require reformulating biological sequence optimization as a continuous optimization, e.g., continuous relaxation (Fu & Levine, 2021; Trabucco et al., 2021; Chen et al., 2022) or mapping discrete designs to a continuous latent space (Yu et al., 2021).

On the other hand, one may consider training deep generative models to learn a distribution over high-scoring designs (Kumar & Levine, 2020; Jain et al., 2022). They learn to generate solutions from scratch, which amortizes optimization over the design space.

Contribution In this paper, we propose a novel algorithm, coined bootstrapped training of score conditioned generator (BOOTGEN), for the offline design of biological sequences. Our key idea is to enhance the score-conditioned generator by suggesting a novel variation of the classical ensemble strategy of bootstrapping and aggregating. We train multiple generators using bootstrapped datasets from training and combine them with proxy mod-

*Equal contribution [1]Department of Industrial and System Engineering, KAIST, Korea [2]Graduate Schoof of Artificial Intelligence, Department of Computer Science and Engineering, POSTECH, Korea. Correspondence to: Jinkyoo Park <jinkyoo.park@kaist.ac.kr>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

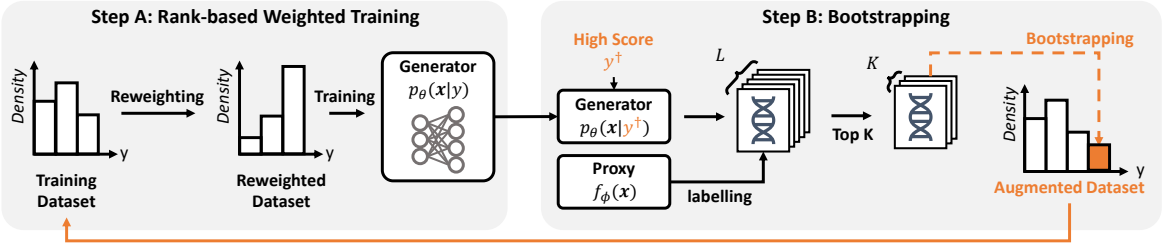


Figure 1: Illustration of the bootstrapped training process for learning score-conditioned generator.

els to create a reliable and diverse sampling solution.

In the bootstrapped training, we aim to align a score-conditioned generator with a proxy function by bootstrapping the training dataset of the generator. To be specific, we repeat multiple stages of (1) training the conditional generator on the training dataset with a focus on high-scoring sequences and (2) augmenting the training dataset using sequences that are sampled from the generator and labeled using the proxy function. Intuitively, our framework improves the score-to-sequence mapping (generator) to be consistent with the sequence-to-score mapping (proxy function), which is typically more accurate.

When training the score-conditioned generator, we assign high rank-based weights (Tripp et al., 2020) to high-scoring sequences. Sequences that are highly ranked among the training dataset are more frequently sampled to train the generator. This leads to shifting the training distribution towards an accurate generation of high-scoring samples. Compared with the value-based weighting scheme previously proposed by Kumar & Levine (2020), the rank-based weighting scheme is more robust to the change of training dataset from bootstrapping.

To further boost the performance of our algorithm, we propose two post-processing processes after the training: filtering and diversity aggregation (DA). The filtering process aims to filter samples from generators using the proxy function to gather samples with cross-agreement between the proxy and generator. On the other hand, DA collects subsamples from multiple generators and combines them into complete samples. DA enables diverse decision-making with reduced variance in generating quality, as it collects samples from multiple bootstrapped generators.

2. BootGen

Problem definition We are interested in optimizing a biological sequence $\boldsymbol{x}$ to maximize a given score function $f(\boldsymbol{x})$. We consider an offline setting where, during optimization, we do not have access to the score function $f(\boldsymbol{x})$. Instead, we optimize the biological sequences using a static dataset $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ consisting of offline queries $y_n = f(\boldsymbol{x}_n)$ to the score function. Finally, we consider evaluating a set of sequences $\{\boldsymbol{x}_m\}_{m=1}^M$ as an output of offline design algorithms.

Overview of BootGen We first provide a high-level description of our bootstrapped training of score-conditioned generator, coined BootGen. Our key idea is to align the score-conditioned generation with a proxy model via bootstrapped training (i.e., we train the generator on sequences labeled using the proxy model) and aggregate the decisions over multiple generators and proxies for reliable and diverse sampling of solutions.

Our BootGen first initializes a training dataset $\mathcal{D}_{\text{tr}}$ as the offline dataset $\mathcal{D}$ and then repeats the following steps:

A. BootGen optimizes the score-conditioned generator $p_\theta(\boldsymbol{x}|y)$ using the training dataset $\mathcal{D}_{\text{tr}}$. During training, it assigns rank-based weights to each sequence for the generator to focus on high scores samples.

B. BootGen bootstraps the training dataset $\mathcal{D}_{\text{tr}}$ using samples from the generator $p_\theta(\boldsymbol{x}|y^\dagger)$ conditioned on the desired score $y^\dagger$. It uses a proxy $f_\phi(\boldsymbol{x}) \approx f(\boldsymbol{x})$ of the score function to label the new samples.

After BootGen training for multiple score-conditioned generators $p_{\theta_1}(\boldsymbol{x}|y), ..., p_{\theta_n}(\boldsymbol{x}|y)$, we aggregate samples from the generators with filtering of proxy score function f_ϕ to generate diverse and reliable samples (see Algorithm 1 and Algorithm 2 for psuedo code).

2.1. Rank-based Weighted Training

Here, we introduce our framework to train the score-conditioned generator. Our algorithm aims to train the score-conditioned generator with more focus on generating high-scoring designs. Such a goal is helpful for bootstrapping and evaluation of our framework, where we query the generator conditioned on a high score.

Given a training dataset $\mathcal{D}_{\text{tr}}$, our BootGen minimizes the following loss function:

$$\mathcal{L}(\theta) := -\sum_{(\boldsymbol{x},y)\in\mathcal{D}_{\text{tr}}} w(y,\mathcal{D}_{\text{tr}}) \log p_\theta(\boldsymbol{x}|y),$$

$$w(y,\mathcal{D}_{\text{tr}}) = \frac{(k|\mathcal{D}_{\text{tr}}| + \text{rank}(y,\mathcal{D}_{\text{tr}}))^{-1}}{\sum_{(\boldsymbol{x},y)\in\mathcal{D}_{\text{tr}}}(k|\mathcal{D}_{\text{tr}}| + \text{rank}(y,\mathcal{D}_{\text{tr}}))^{-1}}.$$

where $w(y,\mathcal{D}_{\text{tr}})$ is the score-wise rank-based weight (Tripp et al., 2020). Here, k is a weight-shifting factor, and

$\mathrm{rank}(y, \mathcal{D}_{\text{tr}})$ denotes the relative ranking of a score y with respect to the set of scores in the dataset $\mathcal{D}_{\text{tr}}$. We note that a small weight-shifting factor k assigns high weights to high-scoring samples. For mini-batch training of the score-conditioned generator, we approximate the loss function $\mathcal{L}(\theta)$ via sampling with probability $w(y, \mathcal{D}_{\text{tr}})$ for each sample $(\boldsymbol{x}, y)$.

We note that Tripp et al. (2020) proposed the rank-based weighting scheme for training unconditional generators to solve online design problems. At a high level, the weighting scheme guides the generator to focus more on generating high-scoring samples. Compared to weights that are proportional to scores (Kumar & Levine, 2020), using the rank-based weights promotes the training to be more robust against outliers, e.g., samples with abnormally high weights. To be specific, the weighting factor $w(y, \mathcal{D})$ is less affected by outliers due to its upper bound that is achieved when $\mathrm{rank}(y, \mathcal{D}) = 1$.

2.2. Bootstrapping

Next, we introduce our bootstrapping strategy to augment a training dataset with high-scoring samples that are collected from the score-conditioned generator and labeled using a proxy model. Our key idea is to enlarge the dataset so that the score-conditioned generation is consistent with predictions of the proxy model, in particular for the high-scoring samples. This enables self-training by utilizing the extrapolation capabilities of the generator and allows the proxy model to transfer its knowledge to the score-conditioned generation process.

We first generate a set of samples $\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_L^*$ from the generator $p_\theta(\boldsymbol{x}|y^\dagger)$ conditioned on the desired score $y^\dagger$ [1] Then we compute the corresponding labels $y_1^*, \ldots, y_L^*$ using the proxy model, i.e., we set $y_\ell = f_\phi(\boldsymbol{x}_\ell)$ for $\ell = 1, \ldots, L$. Finally, we augment the training dataset using the set of top-K samples $\mathcal{D}_{\text{aug}}$ with respect to the proxy model, i.e., we set $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{aug}}$ as the new training dataset $\mathcal{D}_{\text{tr}}$.

2.3. Aggregation Strategy for Sample Generation

Here, we introduce additional post-hoc aggregation strategies that can be used to further boost the quality of samples from our generator. See Algorithm 2 for a detailed process.

Filtering. We follow Kumar & Levine (2020) to exploit the knowledge of the proxy function for filtering high-scoring samples from the generator. To be specific, when evaluating our model, we sample a set of candidate solutions and select the top samples with respect to the proxy function.

Diverse aggregation. To enhance the diversity of candidate samples while maintaining reliable generating performances with low variance, we gather cross-aggregated samples from multiple score-conditioned generators. These generators are independently trained using our proposed bootstrapped training approach. Since each bootstrapped training process introduces high randomness due to varying training datasets, combining the generative spaces of multiple generators yields a more diverse space compared to a single generator.

Moreover, this process helps reduce the variance in generating quality. By creating ensemble candidate samples from multiple generators, we ensure stability and mitigate the risk of potential failure cases caused by adversarial samples. These samples may receive high scores from the proxy function but have low actual scores. This approach resembles the classical ensemble strategy known as "bagging," which aggregates noisy bootstrapped samples from decision trees to reduce variances.

3. Experiments

We present experimental results on six representative biological sequence design tasks to verify the effectiveness of the proposed method. We also conduct ablation studies to verify the effectiveness of each component in our method. For training, we use a single GPU of NVIDIA A100, where the training time of one generator spends approximately 10 minutes. See Appendix A for detailed settings and implementations.

3.1. Experimental Setting

Tasks. We evaluate an offline design algorithm by (1) training it on an offline dataset and (2) using it to generate 128 samples for high scores. We measure the 50th percentile and 100th percentile scores of the generated samples. All the results are measured using eight independent random seeds.

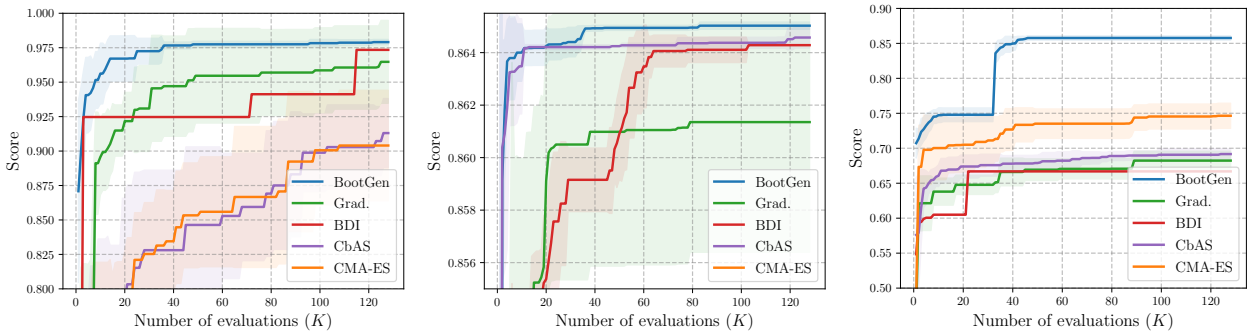
We consider six biological sequence design tasks: green fluorescent protein (GFP), DNA optimization for expression level on untranslated region (UTR), DNA optimization tasks for transcription factor binding (TFBind8), and three RNA optimization tasks for transcription factor binding (RNA-Binding-A, RNA-Binding-B, and RNA-Binding-C). The scores of the biological sequences range in $[0, 1]$. We report the statistics of the offline datasets used for each task in Table 2. We also provide a detailed description of the tasks in Appendix A.1.

Baselines We compare our BootGen with the following baselines: gradient ascent with respect to a proxy score model (Trabucco et al., 2022, Grad.), REINFORCE (Williams, 1992), Bayesian optimization quasi-expected-improvement (Wilson et al., 2017, BO-qEI), covariance

[1] Following (Chen et al., 2022), we assume that we know the maximum score of the task.

Table 1: Experimental results on 100th percentile scores. The mean value is reported for 8 independent solution generations. The 50th stands for the 50th percentile score. The 100th stands for the 100th percentile score. The best-scored value is marked in bold.

Method	RNA-A		RNA-B		RNA-C		TFBind8		GFP		UTR		Avg.	
	50th.	100th.	50th.	100th.	50th.	100th.	50th.	100th.	50th.	100th.	50th.	100th.	50th.	100th.
REINFORCE	0.159	0.462	0.162	0.437	0.177	0.463	0.450	0.936	0.845	**0.865**	0.575	0.685	0.395	0.643
CMA-ES	0.558	0.841	0.531	0.822	0.535	0.803	0.526	0.904	0.047	0.055	0.497	0.737	0.449	0.694
BO-qEI	0.389	0.724	0.397	0.729	0.391	0.707	0.439	0.798	0.246	0.254	0.571	0.684	0.406	0.649
CbAS	0.246	0.541	0.267	0.647	0.281	0.644	0.467	0.913	0.852	**0.865**	0.566	0.692	0.447	0.717
Auto. CbAS	0.241	0.524	0.237	0.562	0.193	0.495	0.413	0.890	0.847	**0.865**	0.563	0.693	0.420	0.672
MIN	0.146	0.376	0.143	0.374	0.174	0.404	0.417	0.892	0.830	**0.865**	0.586	0.691	0.383	0.600
Grad	0.473	0.821	0.462	0.720	0.393	0.688	0.513	0.965	0.763	0.862	0.611	0.682	0.531	0.792
COMs	0.172	0.403	0.184	0.393	0.228	0.494	0.512	0.945	0.737	0.861	0.608	0.699	0.407	0.633
GFN-AL	0.312	0.630	0.300	0.677	0.324	0.623	0.538	0.956	0.051	0.059	0.597	0.695	0.354	0.607
BDI	0.411	0.700	0.308	0.560	0.345	0.632	0.595	0.973	0.837	0.864	0.527	0.667	0.504	0.733
BootGen	**0.707**	**0.902**	**0.717**	**0.931**	**0.596**	**0.831**	**0.833**	**0.979**	**0.853**	**0.865**	**0.701**	**0.865**	**0.731**	**0.895**



(a) TFBind8 (b) GFP (c) UTR

Figure 2: Evaluation-performance graph to compare with representative offline biological design baselines. The number of evaluations $K \in [1, 128]$ stands for the number of candidate designs to be evaluated by the Oracle score function. The average value and standard deviation error bar for 8 independent runs are reported. Our method outperforms other baselines at every task for almost all K.

matrix adaptation evolution strategy (Hansen, 2006, CMA-ES), conditioning by adaptative sampling (Brookes et al., 2019, CbAS), autofocused CbAS (Fannjiang & Listgarten, 2020, Auto. CbAS), model inversion network (Kumar & Levine, 2020, MIN), where these are in the official design bench (Trabucco et al., 2022). We compare with additional baselines of conservative objective models (Trabucco et al., 2021, COMs), generative flow network for active learning (Jain et al., 2022, GFN-AL) and bidirectional learning (Chen et al., 2022, BDI).

3.2. Performance Evaluation

In Table 1, we report the performance of our BootGen along with other baselines. One can observe how our BootGen consistently outperforms the considered baselines across all six tasks.

For TFbind8, which has a relatively small search space (4^8), having high performances on the 100th percentile is relatively easy. Indeed, the classical method of CMA-ES and Grad. gave pretty good performances. However, for the 50th percentile score, which is a metric for measuring the method's reliability, BDI outperformed previous baselines by a large margin. Our method outperformed even BDI and achieved an overwhelming score.

For higher dimensional tasks of UTR, even the 50th percentile score of BootGen outperforms the 100th percentile score of other baselines by a large margin. We note that our bootstrapping strategies and aggregation strategy greatly contributed to improving performances on UTR. For additional tasks of RNA, we achieved the best score for both the 50th percentile and the 100th percentile. This result verifies that our method is task-expandable.

3.3. Additional Results

Varying the evaluation budget. We validate BootGen across multiple evaluation budgets by sampling diverse datasets. As depicted in Fig. 3, BootGen consistently outperforms all other baselines, as demonstrated by its Pareto frontier. For complete results, refer to Appendix C.

Diversity and novelty. We validate multi-objectivity of diversity and novelty of generated samples; see Appendix D

Ablation studies. We provide ablation studies; see Appendix E for overall ablations, and Appendix G for ablation of rank-based weighting.

Calibration model. We suggest experiments of hyperparameter tuning without the score function $f(\boldsymbol{x})$; see Appendix F.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2022R1A6A1A03052954).

References

Angermueller, C., Dohan, D., Belanger, D., Deshpande, R., Murphy, K., and Colwell, L. Model-based reinforcement learning for biological sequence design. In *International conference on learning representations*, 2019.

Barrera, L. A., Vedenko, A., Kurland, J. V., Rogers, J. M., Gisselbrecht, S. S., Rossin, E. J., Woodard, J., Mariani, L., Kock, K. H., Inukai, S., et al. Survey of variation in human transcription factors reveals prevalent dna binding changes. *Science*, 351(6280):1450–1454, 2016.

Brookes, D., Park, H., and Listgarten, J. Conditioning by adaptive sampling for robust design. In *International conference on machine learning*, pp. 773–782. PMLR, 2019.

Brookes, D. H. and Listgarten, J. Design by adaptive sampling. *arXiv preprint arXiv:1810.03714*, 2018.

Chen, C., Zhang, Y., Fu, J., Liu, X., and Coates, M. Bidirectional learning for offline infinite-width model-based optimization. In *Advances in Neural Information Processing Systems*, 2022.

Fannjiang, C. and Listgarten, J. Autofocused oracles for model-based design. *Advances in Neural Information Processing Systems*, 33:12945–12956, 2020.

Fu, J. and Levine, S. Offline model-based optimization via normalized maximum likelihood estimation. *arXiv preprint arXiv:2102.07970*, 2021.

Haldar, R. and Mukhopadhyay, D. Levenshtein distance technique in dictionary lookup methods: An improved approach. *arXiv preprint arXiv:1101.1232*, 2011.

Hansen, N. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation*, pp. 75–102, 2006.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jain, M., Bengio, E., Hernandez-Garcia, A., Rector-Brooks, J., Dossou, B. F., Ekbote, C. A., Fu, J., Zhang, T., Kilgour, M., Zhang, D., et al. Biological sequence design with gflownets. In *International Conference on Machine Learning*, pp. 9786–9801. PMLR, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kumar, A. and Levine, S. Model inversion networks for model-based optimization. *Advances in Neural Information Processing Systems*, 33:5126–5137, 2020.

Lorenz, R., Bernhart, S. H., Höner zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P. F., and Hofacker, I. L. Viennarna package 2.0. *Algorithms for molecular biology*, 6(1):1–14, 2011.

Ren, Z., Li, J., Ding, F., Zhou, Y., Ma, J., and Peng, J. Proximal exploration for model-guided protein sequence design. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18520–18536. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/ren22a.html.

Sample, P. J., Wang, B., Reid, D. W., Presnyak, V., McFadyen, I. J., Morris, D. R., and Seelig, G. Human 5' utr design and variant effect prediction from a massively parallel translation assay. *Nature biotechnology*, 37(7): 803–809, 2019.

Trabucco, B., Kumar, A., Geng, X., and Levine, S. Conservative objective models for effective offline model-based optimization. In *International Conference on Machine Learning*, pp. 10358–10368. PMLR, 2021.

Trabucco, B., Geng, X., Kumar, A., and Levine, S. Design-bench: Benchmarks for data-driven offline model-based optimization. *arXiv preprint arXiv:2202.08450*, 2022.

Tripp, A., Daxberger, E., and Hernández-Lobato, J. M. Sample-efficient optimization in the latent space of deep generative models via weighted retraining. *Advances in Neural Information Processing Systems*, 33:11259–11272, 2020.

Wang, H., Sakhadeo, A., White, A. M., Bell, J. M., Liu, V., Zhao, X., Liu, P., Kozuno, T., Fyshe, A., and White, M. No more pesky hyperparameters: Offline hyperparameter tuning for RL. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=AiOUi3440V.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Wilson, J. T., Moriconi, R., Hutter, F., and Deisenroth, M. P. The reparameterization trick for acquisition functions. *arXiv preprint arXiv:1712.00424*, 2017.

Yu, S., Ahn, S., Song, L., and Shin, J. Roma: Robust model adaptation for offline model-based optimization. *Advances in Neural Information Processing Systems*, 34: 4619–4631, 2021.

Zimmer, M. Green fluorescent protein (GFP): applications, structure, and related photophysical behavior. *Chemical reviews*, 102(3):759–782, 2002.

A. Additional Experimental Settings

Table 2: Details of the offline datasets. We let $|\mathcal{X}|$ and $|\mathcal{D}|$ denote the sizes of the search space and the offline dataset, respectively.

	Seq. Length	Vocab size	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{D}\rvert$
GFP	20	237	20^{237}	5,000
UTR	50	4	50^4	140,000
TFBind8	8	8	4^8	32,898
RNA-Binding	14	4	4^{14}	5,000

A.1. Datasets

- **GFP** (Sample et al., 2019) is a task to optimize a protein sequence of length 237 consisting of one of 20 amino acids, i.e., the search space is 20^{237}. Its objective is to find a protein with high fluorescence. Following Trabucco et al. (2022), we prepare the offline dataset using 5000 samples with 50 to 60 percentile scores in the original data.

- **UTR** (Sample et al., 2019) is a task to optimize a DNA sequence of length 50 consisting of one of four nucleobases: adenine (A), guanine (G), cytosine (C), thymine (T). Its objective is to maximize the expression level of the corresponding 5'UTR region. For the construction of the offline dataset $\mathcal{D}$, following Trabucco et al. (2022), we provide samples with scores under the 50th percentile data of $140,000$ examples.

- **TFBind8** (Barrera et al., 2016) is a task that optimizes DNA similar to the UTR. The objective is to find a length 8 sequence to maximize the binding activity with human transcription factors. For the offline dataset $\mathcal{D}$, we provide under 50th percentile data of 32,898 examples following Trabucco et al. (2022).

- **RNA-Binding** (Lorenz et al., 2011) is a task that optimizes RNA, a sequence that contains four vocab words of nucleobases: adenine (A), uracil (U), cytosine (C), and guanine (G). The objective is to find a length 14 sequence to maximize the binding activity with the target transcription factor. We present three target transcriptions of RNA termed RNA-Binding-A (for L14 RNA1), RNA-Binding-B (for L14 RNA2), and RNA-Binding-C (for L14 RNA3). We provide under 0.12 scored data for the offline dataset $\mathcal{D}$ among randomly generated 5,000 sequences using opensource code [2].

A.2. Implementation of Baselines

This section provides a detailed implementation of baselines of offline biological sequence design.

Baselines from Design Bench (Trabucco et al., 2022). Most baselines are from the offline model-based optimization (MBO) benchmark called design-bench (Trabucco et al., 2022). The design bench contains biological sequence tasks of the GFP, UTR, and TFbind8, where it contains baselines of REINFORCE, CMA-ES (Hansen, 2006), BO-qEI (Wilson et al., 2017), CbAS (Brookes et al., 2019), Auto. Cbas (Fannjiang & Listgarten, 2020), MIN (Kumar & Levine, 2020), gradient ascent (Grad.), and COMS (Trabucco et al., 2021). We reproduce them by following the official source code [3]. For the RNA tasks, we follow hyperparameters of TFBind8 as the number of vocab are same as 4 and the sequence length is similar where the TFBind8 has length 8 and RNA tasks have length 14 as our method follows the same.

BDI (Chen et al., 2022). For BDI, we follow hyperparameter setting at the paper (Chen et al., 2022) and implementation at the opensource code [4]. For RNA tasks, we follow the hyperparameter for TFBind8 tasks, as our method follows the same.

GFN-AL (Jain et al., 2022). For GFN-AL we follow hyperparameters setting at the paper (Jain et al., 2022) and implementation on open-source code[5]. Because they only reported the TFbind8 and the GFP tasks, we use the hyperparameter of the GFP for the UTR tasks hyperparameter of the TFBind8 for RNA tasks, as our method follows the same.

[2] https://github.com/samsinai/FLEXS
[3] https://github.com/brandontrabucco/design-baselines
[4] https://github.com/GGchen1997/BDI
[5] https://github.com/MJ10/BioSeq-GFN-AL

A.3. Implementation of BOOTGEN

We parameterize the conditional distribution $p_\theta(x_t|\boldsymbol{x}_{1:t-1}, y)$ using a 2-layer long short-term memory (Hochreiter & Schmidhuber, 1997, LSTM) network with 512 hidden dimensions. The condition y is injected into the LSTM using a linear projection layer. We parameterize the proxy model using a multi-layer perceptron (MLP) with 2048 hidden dimensions and a sigmoid activation function. Our parameterization is consistent across all the tasks. We also note the importance of the desired score $y^\dagger$ to condition during bootstrapping and evaluation. In this regard, we set it as the maximum score that is achievable for the given problem, i.e., we set $y^\dagger = 1$. We assume that such a value is known following (Chen et al., 2022).

A.4. Hyperparameters

Training. We give consistency hyperparameters for all tasks except the learning rate. We set the generator's learning rate to 10^{-5} for short-length tasks (lengths 8 and 14) of TFBind and RNA tasks and 5×10^{-5} for longer-length tasks (lengths 50 and 237) of UTR and GFP. We trained the generator with 12,500 steps before bootstrapping. Bootstrapping is applied with $2,500$ in additional steps. The batch size of training is 256. We set the weighting parameter $k = 10^{-2}$. Note that we early stopped the generator iteration of GFP with the $3,000$ step based on monitoring the calibration model of **??**. For bootstrapping, the generator samples a 2 candidates every 5 steps. For Top-K sampling at the bootstrapping, we sample with $L = 1,000$ and select the Top 2 samples to augment the training dataset.

Testing. For filtering, we generated $M = 1,280$ candidate samples and collected the Top-K samples where $K = 128$ based on the proxy score. For diverse aggregation, we collect $K = 16$ samples from 8 generators, making total 128 samples.

Proxy model. For the proxy model, we applied a weight regularization of 10^{-4}, set the learning rate to 10^{-4}, and used a dropout rate of 0.1. We used early stopping with a tolerance of 5 and a train/validate ratio of 9:1 following Jain et al. (2022). We used the Adam optimizer (Kingma & Ba, 2014) for the training generator, proxy, and calibration model.

B. Algorithm for BOOTGEN

This section provides algorithmic pseudo code for BOOTGEN This has a training process of multiple generators and a sampling process to obtain candidate samples from the trained multiple generators.

Algorithm 1 Bootrapped Training of Score-conditioned generators

Input: Offline dataset $\mathcal{D} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$.
Update ϕ to minimize $\sum_{(\boldsymbol{x},y)\in\mathcal{D}}(f_\phi(\boldsymbol{x}) - y)^2$.
for $j = 1, \ldots, N_{\text{gen}}$ **do** ▷ Training multiple generaters
  Initialize $\mathcal{D}_{\text{tr}} \leftarrow \mathcal{D}$.
  for $i = 1, \ldots, I$ **do** ▷ Rank-based weighted training
    Update θ_n to maximize $\sum_{(\boldsymbol{x},y)\in\mathcal{D}_{\text{tr}}} w(y, \mathcal{D}_{\text{tr}}) \log p_{\theta_j}(\boldsymbol{x}|y)$.
    Sample $\boldsymbol{x}_\ell^* \sim p_{\theta_n}(\boldsymbol{x}|y^\dagger)$ for $\ell = 1, \ldots, L$. ▷ Bootstrapping
    Set $y_\ell^* \leftarrow f_\phi(x_\ell^*)$ for $\ell = 1, \ldots, L$.
    Set $\mathcal{D}_{\text{aug}}$ as top-K scoring samples in $\{\boldsymbol{x}_\ell^*, y_\ell^*\}_{\ell=1}^L$.
    Set $\mathcal{D}_{\text{tr}} \leftarrow \mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{aug}}$.
  end for
end for
Output: trained score-conditioned generators $p_{\theta_1}(\boldsymbol{x}|y), ..., p_{\theta_N}(\boldsymbol{x}|y)$.

Algorithm 2 Aggregation Strategy for Sample Generation

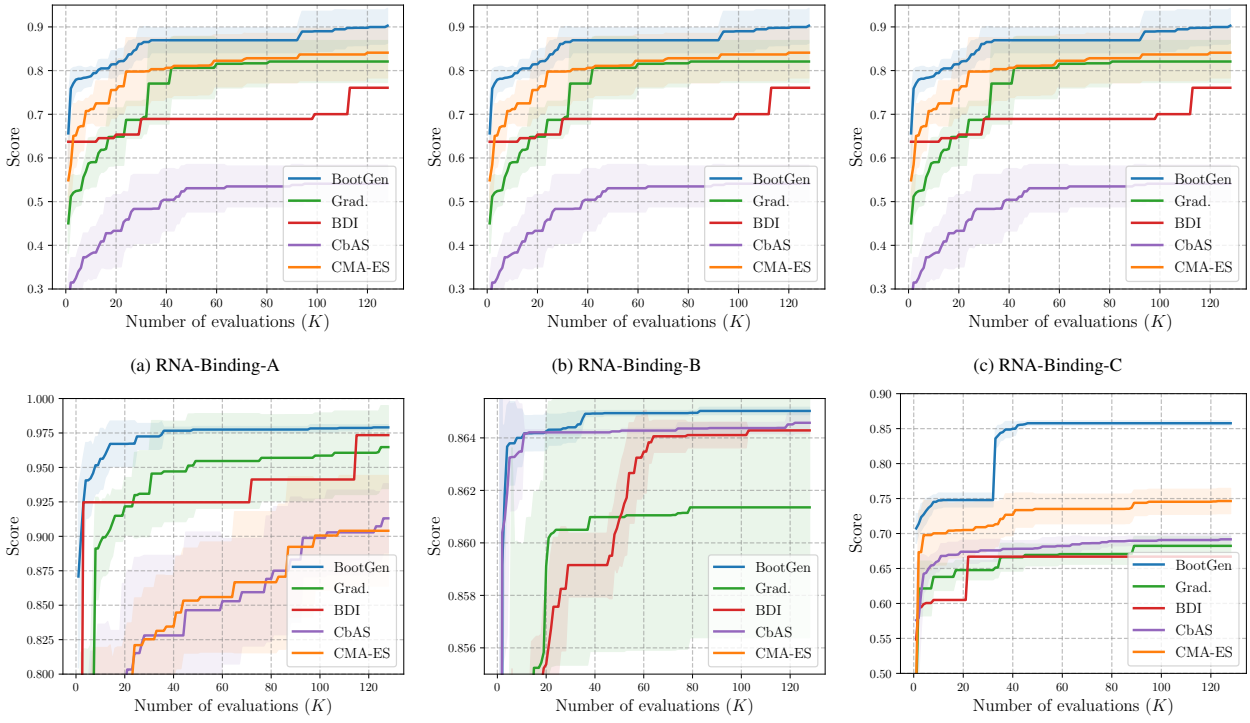
Input: Trained score-conditioned generators $p_{\theta_1}(\boldsymbol{x}|y), ..., p_{\theta_{N_{\text{gen}}}}(\boldsymbol{x}|y)$, and trained proxy function $f_\phi(\boldsymbol{x})$.
Initialize $\mathcal{D}_{\text{samples}} \leftarrow \emptyset$.
for $i = 1, \ldots, N_{\text{gen}}$ **do**
  Sample $\boldsymbol{x}_m^* \sim p_{\theta_i}(\boldsymbol{x}|y^\dagger)$ for $m \in [M]$.
  Set $y_m^* \leftarrow f_\phi(x_m^*)$ for $m \in [M]$.
  Set $\mathcal{D}_{\text{sub-samples}}$ as Top-K scoring samples in $\{\boldsymbol{x}_m^*, y_m^*\}_{m=1}^M$. ▷ Filtering
  Set $\mathcal{D}_{\text{samples}} \leftarrow \mathcal{D}_{\text{samples}} \cup \mathcal{D}_{\text{sub-samples}}$ ▷ Diversity Aggregation
end for
Output: $\mathcal{D}_{\text{samples}}$.

C. Varying the evaluation budget

In real-world scenarios, there may be situations where only a few samples can be evaluated due to the expensive score function. For example, in an extreme scenario, for the clinical trial of a new protein drug, there may be only one or two chances to be evaluated. As we measure the 50th percentile and 100th percentile score among 128 samples following the design-bench (Trabucco et al., 2022) at Table 1, we also provide a 100th percentile score report at the fewer samples from 1 sample to the 128 samples to evaluate the model's robustness on the low-budget evaluation scenarios.

To account for this, we also provide a budget-performance graph that compares the performance of our model to the baselines using different numbers of evaluations. This allows us to observe the trade-off between performance and the number of samples generated. Note that we select baselines as the Top 5 methods in terms of average percentile 100 scores reported at Table 1.

As shown in Fig. 3, our method outperforms every baseline for almost every evaluation budget. For the UTR task, our performance on a single evaluation budget gives a better score than the other baselines' scores when they have a budget of 128 evaluations. For RNA tasks, our method with an approximate budget of 30 achieves superior performance compared to other methods with a budget of 128. These results show that our method is the most reliable as its performance is most robust when the evaluation budget is limited.



(a) RNA-Binding-A (b) RNA-Binding-B (c) RNA-Binding-C

(d) TFBind8 (e) GFP (f) UTR

Figure 3: Evaluation-performance graph to compare with representative offline biological design baselines. The number of evaluations $K \in [1, 128]$ stands for the number of candidate designs to be evaluated by the Oracle score function. The average value and standard deviation error bar for 8 independent runs are reported. Our method outperforms other baselines at every task for almost all K.

D. Average Score with Diversity

Table 3: Experimental results on 100th percentile scores (100th Per.), 50th percentile scores (50th Per.), average score (Avg. Score), diversity, and novelty, among 128 samples of UTR task. The mean and standard deviation of 8 independent runs for producing 128 samples is reported. The best-scored value is marked in bold. The lowest standard deviation is marked as the underline. The DA stands for the diverse aggregation strategy.

Methods	100th Per.	50th Per.	Avg. Score	Diversity	Novelty
MIN (Kumar & Levine, 2020)	0.691 ± 0.011	0.587 ± 0.012	0.554 ± 0.010	28.53 ± 0.095	18.32 ± 0.091
CMA-ES (Hansen, 2006)	0.746 ± 0.018	0.498 ± 0.012	0.520 ± 0.013	24.69 ± 0.150	19.95 ± 0.925
Grad. (Trabucco et al., 2022)	0.682 ± 0.013	0.513 ± 0.007	0.521 ± 0.006	25.63 ± 0.615	16.89 ± 0.426
GFN-AL (Jain et al., 2022)	0.700 ± 0.015	0.602 ± 0.014	0.580 ± 0.014	30.89 ± 1.220	20.25 ± 2.272
BootGen w/o DA.	0.729 ± 0.074	0.672 ± 0.082	0.652 ± 0.081	17.83 ± 5.378	20.49 ± 1.904
BootGen w/ DA. (*ours*)	**0.858** ± 0.003	**0.701** ± 0.004	**0.698** ± 0.001	**31.57** ± 0.073	**21.40** ± 0.057

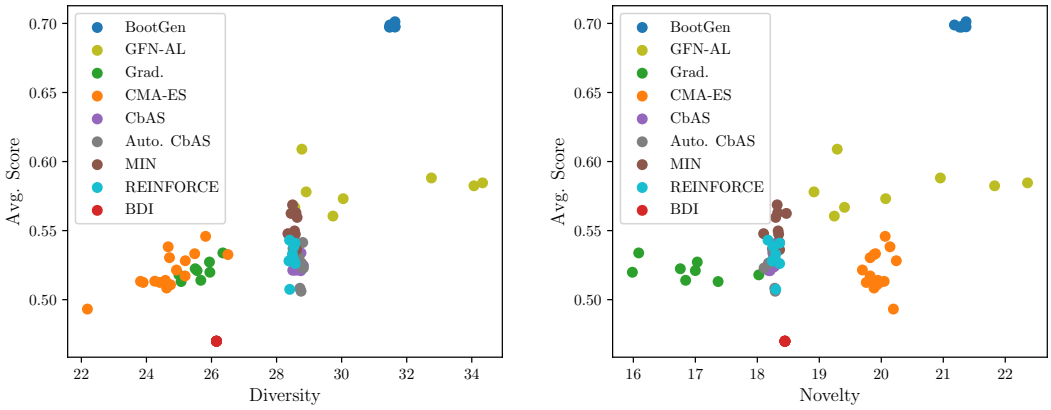


Figure 4: Multi-objectivity comparison of diversity and novelty on the average score for the UTR task. Each datapoint for 8 independent runs is depicted.

For biological sequence design, measuring the diversity and novelty of the generated sequence is also crucial (Jain et al., 2022). Following the evaluation metric of (Jain et al., 2022) we compare the performance of models in terms of diversity and novelty.

Here is measurement of diversity for sampled design dataset $\mathcal{D} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_M\}$ from generator which is average of *Levenshtein distance* (Haldar & Mukhopadhyay, 2011), denoted by $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$, between arbitrary two biological sequences $\boldsymbol{x}_i, \boldsymbol{x}_j$ from the generated design candidates $\mathcal{D}$:

$$\text{Diversity}(\mathcal{D}) \coloneqq \frac{1}{|\mathcal{D}|(|\mathcal{D}|-1)} \sum_{\boldsymbol{x}\in\mathcal{D}} \sum_{\boldsymbol{s}\in\mathcal{D}\setminus\{\boldsymbol{x}\}} d(\boldsymbol{x}, \boldsymbol{s}).$$

Next, we measure the minimum distance from the offline dataset $\mathcal{D}_{\text{offline}}$ which measures the novelty of generated design candidates $\mathcal{D}$ as:

$$\text{Novelty}(\mathcal{D}, \mathcal{D}_{\text{offline}}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}\in\mathcal{D}} \min_{\boldsymbol{s}\in\mathcal{D}_{\text{offline}}} d(\boldsymbol{x}, \boldsymbol{s}).$$

Our method surpasses all baselines, including GFN-AL (Jain et al., 2022), in the UTR task, as evidenced by the Pareto frontier depicted in Table 3 and Fig. 4. Given the highly dimensional nature of the UTR task and its expansive search space, the discovery of novel and diverse candidates appears to be directly related to their average score. This implies that extensive exploration of the high-dimensional space is crucial for improving scores in the UTR task.

It is worth noting that GFN-AL, which is specifically designed to generate diverse, high-quality samples through an exploratory policy, secures a second place for diversity. Although GFN-AL occasionally exhibits better diversity than our

method and achieves a second-place average score, it consistently delivers poor average scores in the GFP and RNA tasks Table 1. This drawback can be attributed to its high explorative policy, which necessitates focused exploration in narrow regions. In contrast, BOOTGEN consistently produces reliable scores across all tasks Table 1.

Diverse aggregation strategy Our diverse aggregation (DA) strategy significantly enhances diversity, novelty, and score variance, as demonstrated in Table 3. This is especially beneficial for the UTR task, which necessitates extensive exploration of a vast solution space, posing a substantial risk to the bootstrapped training process. In this context, certain bootstrapped generators may yield exceedingly high scores, while others may produce low scores due to random exploration scenarios. By employing DA, we combine multiple generators to generate candidate samples, thereby greatly stabilizing the quality of the bootstrapped generator.

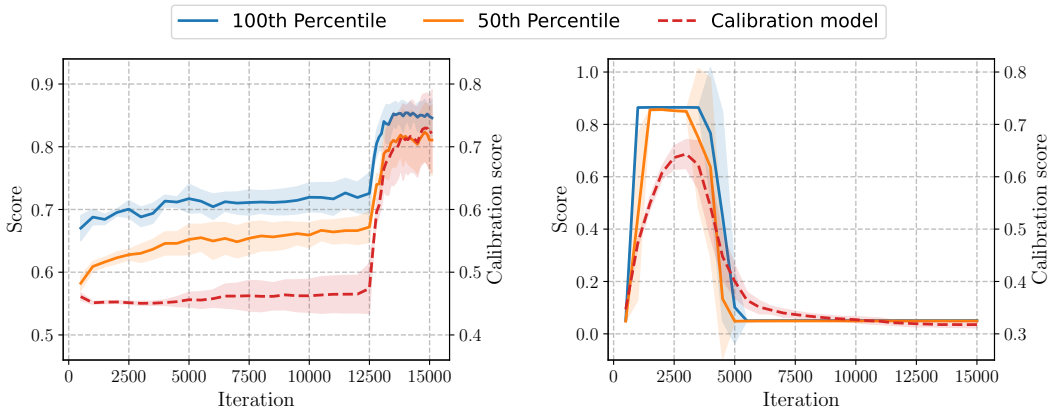
E. Ablation study

The effectiveness of our components, namely rank-based reweighting (RR), bootstrapping (B), and filtering (F), in improving performance is evident in Table 4. Across all tasks, these components consistently contribute to performance enhancements. The bootstrapping process is particularly more beneficial for high-dimensional tasks like UTR and GFP. This correlation is intuitive since high-dimensional tasks require a larger amount of data for effective exploration. The bootstrapped training dataset augmentation facilitates this search process by leveraging proxy knowledge. Additionally, the filtering technique proves to be powerful in improving scores. As we observed from the diverse aggregation and filtering, the ensemble strategy greatly enhances score-conditioned generators.

Table 4: Ablation study for BOOTGEN. The average score among 128 samples is reported. We make 8 independent runs to produce 128 samples where the mean and the standard deviation are reported. For every method, an aggregation strategy is applied by default. The best-scored value is marked in bold. The lowest standard deviation is underlined. The RR stands for rank-based reweighting, the B stands for bootstrapping, and the F stands for filtering.

Components	RNA-A	RNA-B	RNA-C	TFbind8	UTR	GFP
$\emptyset$	0.388 ± 0.007	0.350 ± 0.008	0.394 ± 0.010	0.579 ± 0.010	0.549 ± 0.009	0.457 ± 0.044
{RR}	0.483 ± 0.006	0.468 ± 0.008	0.441 ± 0.010	0.662 ± 0.009	0.586 ± 0.008	0.281 ± 0.031
{RR, B}	0.408 ± 0.009	0.379 ± 0.009	0.417 ± 0.006	0.666 ± 0.009	0.689 ± 0.003	0.470 ± 0.034
{RR, F}	0.576 ± 0.005	0.586 ± 0.004	0.536 ± 0.007	0.833 ± 0.004	0.621 ± 0.003	0.783 ± 0.011
{RR, F, B}	**0.607** ± 0.009	**0.612** ± 0.005	**0.554** ± 0.007	**0.840** ± 0.004	**0.698** ± 0.001	**0.804** ± 0.002

F. Calibration Model



(a) TFbind8 (b) GFP

Figure 5: Calibration model's tendency.

Tuning the hyperparameters of offline design algorithms is challenging due to the lack of access to the true score function. Therefore, existing works have proposed various strategies to circumvent this issue, e.g., choosing a hyperparameter that is transferrable between different tasks (Trabucco et al., 2022) or tuning the hyperparameter based on training statistics (Yu et al., 2021).

In this work, we leverage the calibration function. Inspired by Wang et al. (2022), we train the calibration function on the offline dataset to approximate the true score function similar to the proxy function. Then we use the calibration function to select a score-conditioned model which achieves higher performance with respect to the calibration function. We also choose the number of training steps and early stopping points using the same criterion.

As shown in Fig. 5, the calibration model accurately predicts early stopping points as the GFP task is unstable and has a narrow high score region which gives a high chance to be overfitted into the low-scored region (Table 1 shows that score of GFP is highly polarized). By using the calibration function, we can simply choose an early stopping point for GFP. Note we simply leverage the proxy model as a calibration model with an exact sample training scheme and hyperparameters.

G. Rank-based weighting vs. Value-based weighting

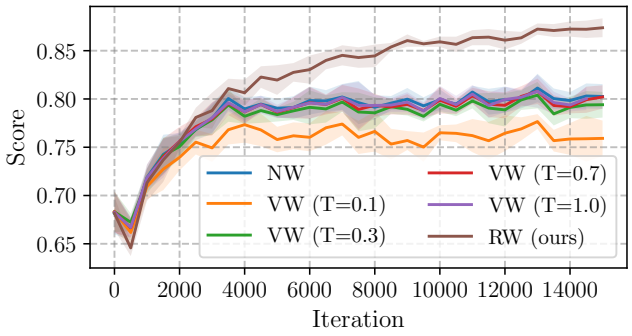


Figure 6: Comparison of rank-based weighting (RW) and value-based weighting (VW) methods. The NW represents the case where no weighting is applied to the training distribution. In the VW case, we explored different weighting temperatures, $T \in \{0.1, 0.3, 0.7, 1.0\}$. The 50th percentile scores of TFBind8 are reported, and the results include a bootstrapping procedure applied from iteration 12,500 to 15,000.

We verify the contribution of the rank-based weighting (RW) scheme compared to ours with no weighting (NW) and the existing value-based weighting (VW) proposed by Kumar & Levine (2020). To implement VW, we set the sample-wise weight proportional to $\exp(|y - y^*|/T)$, where y^* is the maximum score in the training dataset and $T \in \{0.1, 0.3, 0.7, 1.0\}$ is a hyperparameter. As shown in Fig. 6, the results indicate that RW indeed outperforms both NW and VW. This validates our design choice for BootGen.