

This appendix presents a refined version of the supplementary materials originally submitted with the main text. It also includes an additional example of another design for the probabilistic bridge framework, the Ornstein-Uhlenbeck bridges, which is an alternative to the Brownian bridge from the original submission.

## A Task Details

### A.1 Design-Bench Task

Design-Bench [40] is a widely adopted benchmark for evaluating offline black-box optimization algorithms. Table 5 presents the summary of five evaluation tasks in Design-Bench.

Table 5: Overview of tasks in the Design-Bench benchmark. The ground-truth oracle functions for these tasks (unknown to our algorithm) are accessible during evaluation.

Task	Offline Size	Input Dimension	Input Category	Input Type	Oracle
TF Bind 8	32,898	8	4	Discrete	Exact
TF Bind 10	10,000	10	4	Discrete	Exact
Ant Morphology	10,004	60	N/A	Continuous	Exact
D’Kitty Morphology	10,004	56	N/A	Continuous	Exact
RNA-Binding	5000	14	4	Discrete	Exact

**TF Bind 8 and TF Bind 10: DNA Sequence Optimization.** The goals of the **TF Bind 8** and **TF Bind 10** tasks are to discover, respectively, the length-8 and length-10 DNA sequences with the strongest binding affinity to a specific transcription factor (SIX6\_REF\_R1 by default) [2]. In these settings, each DNA candidate is a sequence of nucleotides where each nucleotide has 4 possible categorical values. The Design-Bench benchmark grants access to the full oracle functions for the **TF-Bind-8** and **TF-Bind-10** tasks, which correspond to databases containing exact binding affinities for all possible sequences (i.e.,  $4^8$  and  $4^{10}$  combinations, respectively). Following the evaluation protocol in [40], a fixed subset of sequences is sampled from each oracle and used as the offline dataset for all baselines. In particular, **TF-Bind-8** provides an offline dataset of 32,898 sequences while **TF-Bind-10** provides an offline dataset with 10,000 sequences.

**Ant and D’Kitty Morphology: Robot Morphology Optimization.** These tasks focus on optimizing the physical structure of two simulated robots: (1) Ant from OpenAI Gym [4] and (2) D’Kitty from ROBEL [1]. In **Ant Morphology**, the objective is to find the structure of a quadruped robot to maximize its running speed. In **D’Kitty Morphology** the goal is to find the most effective structure for the D’Kitty robot that enables it to reach a specific target. In particular, each structure candidate specifies the morphology parameters, such as the size, orientation, and placement of limbs, for a robot controller which will be trained using the Soft Actor-Critic algorithm [19]. For the Ant robots, there are 60 morphology parameters. For the D’Kitty robots, there are 56 morphology parameters. The solution quality of each structure candidate is obtained by simulating the corresponding trained controller in the MuJoCo [39] environment. Each simulation runs for 100 steps and the overall solution quality is obtained by averaging simulation results across 16 independent runs.

### A.2 RNA Task

**RNA-Binding** [30]. This is an inverse-folding task whose objective is to optimize RNA sequences of fixed length (14 nucleotides) over the alphabet  $\{A, U, C, G\}$ , which are abbreviations for adenine, uracil, cytosine, and guanine, respectively. In particular, the task is to find a sequence whose predicted minimum-free-energy (MFE) structure maximizes its binding affinity to a given transcription factor. We follow the benchmark setup of Bootgen [24] by considering three target structures **RNA-A** (L14 RNA1), **RNA-B** (L14 RNA2), and **RNA-C** (L14 RNA3). To construct the offline dataset  $D_o$ , we use the FLEXS codebase [34] to generate sequences uniformly at random. The RNAinverse algorithm from ViennaRNA 2.0 is then applied iteratively until 5,000 sequences with minimum free energy (MFE) below 0.12 are obtained, forming the final offline dataset.

---

**Algorithm 1** Synthetic Data Generation via Simulating Gaussian Process (GP) Posteriors

---

1: **Input:** Offline dataset  $D_o = \{\mathbf{x}_i, y_i\}_{i=1}^n$ , number of functions per epoch  $n_e$ , number of points per function  $n_p$ , number of gradient steps  $M$   
2: **Output:** Synthetic dataset  $D_s = \{(\mathbf{X}^-, \mathbf{y}^-), (\mathbf{X}^+, \mathbf{y}^+)\}$   
3:  $\mathcal{D}_s \leftarrow \emptyset$   
4: **for**  $s = 1$  **to**  $n_e$  **do**  
5:   Sample kernel parameters  $\phi_s = (\ell_s, \sigma_s^2)$ :  $\ell_s \sim U(\ell_0 - \delta, \ell_0 + \delta)$ ,  $\sigma_s^2 \sim U(\sigma_0^2 - \delta, \sigma_0^2 + \delta)$   
6:   Compute the mean function  $\bar{g}_{\phi_s}$  of the posterior Gaussian process via Eq. 11  
7:   Sample a subset of  $n_p$  points from offline data:  $\mathcal{D}_0 \subset D_o$  where  $|\mathcal{D}_0| = n_p$   
8:   Compute the set of low-value designs  $\mathbf{X}_s^-$  using Eq. 12  
9:   Compute the set of high-value designs  $\mathbf{X}_s^+$  using Eq. 13  
10:   Compute corresponding low and high scores:  $\mathbf{y}_s^- = \bar{g}_{\phi_s}(\mathbf{X}_s^-)$ ,  $\mathbf{y}_s^+ = \bar{g}_{\phi_s}(\mathbf{X}_s^+)$   
11:    $D_s \leftarrow D_s \cup \{(\mathbf{X}_s^-, \mathbf{y}_s^-), (\mathbf{X}_s^+, \mathbf{y}_s^+)\}$   
12: **end for**  
13: **Return**  $D_s$

---

## 525 B Detailed Algorithmic Description and Implementation of ROOT

### 526 B.1 Generating Synthetic Data with GP

Table 6: Hyperparameters for the synthetic data generation of **ROOT**.

Hyperparameter	Value
$\ell_0, \sigma_0^2$	1.0 (continuous) 6.25 (discrete)
$\delta$	0.25
Step size ( $\eta$ )	0.001 (continuous) 0.05 (discrete)
Number of gradient steps ( $M$ )	100
Threshold ( $\tau$ )	0.001

527 To generate a diverse range of synthetic functions which are sufficiently similar to the oracle, we  
528 first sample the kernel parameters  $\ell_s$  (lengthscale) and  $\sigma_s^2$  (variance) uniformly from the ranges  
529  $[\ell_0 - \delta, \ell_0 + \delta]$  and  $[\sigma_0^2 - \delta, \sigma_0^2 + \delta]$ , where  $\ell_0$ ,  $\sigma_0^2$  and  $\delta$  are fixed initial hyperparameters, as reported  
530 in Table 6. After sampling, we compute the mean function of the Gaussian process posterior based on  
531 the offline data. We then sample  $n_p$  points from the offline data and perform  $M = 100$  gradient ascent  
532 and gradient descent steps with a step size  $\eta$  (more details are provided in Section 3.3). To enhance  
533 the quality of our synthetic data, we filter out any pair of low- and high-value points  $(\mathbf{x}^-, y^-)$  and  
534  $(\mathbf{x}^+, y^+)$  whose difference between  $y^+$  and  $y^-$  is smaller than a threshold  $\tau$ . All key hyperparameters  
535 for this process are listed in Table 6. The pseudocode of the algorithm is presented in Algorithm 1.

### 536 B.2 Learning the Probabilistic Bridge Model

537 This section provides further details regarding the implementation of the learning loss in Eq. 8 with  
538 respect to its Brownian bridge instantiation in Section 3.2. In particular, following the formulation of  
539  $q(\mathbf{x}_t \mid \mathbf{x}_0, \mathbf{x}_T)$  in Eq. 10, we can express  $\mathbf{x}_t$

$$\mathbf{x}_t = \left(1 - \frac{t}{T}\right) \mathbf{x}_0 + \frac{t}{T} \mathbf{x}_T + \sqrt{\kappa_{t,t}} \epsilon_t \quad (15)$$

540 where  $\epsilon_t \sim \mathbb{N}(0, \mathbf{I})$  and  $\kappa_{t,t} = 2(m_t - m_t^2)$  with  $m_t = t/T$ . This reproduces the Brownian Bridge  
541 Diffusion Model [29]. Following the derivation in [29], we obtain:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_T) = \mathbb{N}(\boldsymbol{\mu}(\mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_T), \tilde{\kappa}_{t-1} \cdot \mathbf{I}),$$

542 where the mean is

$$\boldsymbol{\mu}(\mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_T) = u_t \cdot \mathbf{x}_t + v_t \cdot \mathbf{x}_T + w_t \cdot (m_t(\mathbf{x}_T - \mathbf{x}_0) + \sqrt{\kappa_{t,t}} \epsilon), \quad (16)$$

---

**Algorithm 2** Learning the Probabilistic Bridge and Simulation for Offline Optimization (**ROOT**)

---

```

1: Input: Offline dataset  $D_o = \{\mathbf{x}_i, y_i\}_{i=1}^n$ , number of epochs  $E$ , number of diffusion steps  $T$ ,
   scale  $\alpha$ , best objective value  $y_*$ , conditional dropout probability  $\rho$ , number of iterations  $I$ 
2: Output: High-value candidate  $\mathbf{x}_0^*$ 
3:
4: Learning Probabilistic Bridge Phase:
5: Initialize model parameters  $\theta$ 
6: for  $e = 1$  to  $E$  do
7:   Generate synthetic dataset  $D_s$  from Algorithm 1
8:   for  $i = 1$  to  $I$  do
9:     Sample  $\{\mathbf{x}_T, y_T, \mathbf{x}_0, y_0\} \sim \mathcal{D}_s = \{\mathbf{X}^-, \mathbf{y}^-, \mathbf{X}^+, \mathbf{y}^+\}$ 
10:    Sample timestep  $t \sim \text{Uniform}(1, T)$ ,  $\gamma \sim \text{Ber}(\rho)$ ,  $y = [y_T, y_0]$ 
11:    Forward diffusion:  $\mathbf{x}_t = (1 - m_t) \cdot \mathbf{x}_0 + m_t \cdot \mathbf{x}_T + \sqrt{\kappa_{t,t}} \epsilon$  where  $\epsilon \sim \mathbb{N}(0, \mathbf{I})$ 
12:    Gradient descent step:  $\nabla_{\theta} \|m_t(\mathbf{x}_T - \mathbf{x}_0) + \sqrt{\kappa_{t,t}} \epsilon - \epsilon_{\theta}(\mathbf{x}_t, t, (1 - \gamma) \cdot y + \gamma \cdot \emptyset)\|^2$  (Eq. 24)
13:    end for
14:  end for
15:
16: Simulation Phase:
17:  $\{\mathbf{x}_T, y_T\} \leftarrow$  128 best designs in  $D_o$ ,  $y = [y_T, \alpha \cdot y_*]$ 
18: for  $t = T$  to 1 do
19:    $\mathbf{z} \sim \mathbb{N}(0, \mathbf{I})$  if  $t > 0$  else  $\mathbf{z} = 0$ 
20:   Compute  $\epsilon_{\theta}(\mathbf{x}_t, t, y)$  via Eq. 25
21:    $\mathbf{x}_{t-1} = u_t \cdot \mathbf{x}_t + v_t \cdot \mathbf{x}_T + w_t \cdot \epsilon_{\theta}(\mathbf{x}_t, t, y) + \sqrt{\tilde{\kappa}_{t-1}} \cdot \mathbf{z}$ 
22: end for
23:
24: Return: high-value design  $\mathbf{x}_0^* = \mathbf{x}_0$ 

```

---

543 and the variance is

$$\tilde{\kappa}_{t-1} = \left( \kappa_{t,t} - \kappa_{t-1,t-1} \cdot \frac{(1 - m_t)^2}{(1 - m_{t-1})^2} \right) \cdot \frac{\kappa_{t-1,t-1}}{\kappa_{t,t}}. \quad (17)$$

544 Furthermore, the coefficients for the above mean function were derived as

$$u_t = \frac{\kappa_{t-1,t-1}}{\kappa_{t,t}} \cdot \frac{1 - m_t}{1 - m_{t-1}} + \frac{\kappa_{t,t} - \kappa_{t-1,t-1} \cdot \frac{(1 - m_t)^2}{(1 - m_{t-1})^2}}{\kappa_{t,t}} \cdot (1 - m_{t-1}), \quad (18)$$

$$v_t = m_{t-1} - m_t \cdot \frac{1 - m_t}{1 - m_{t-1}} \cdot \frac{\kappa_{t-1,t-1}}{\kappa_{t,t}}, \quad (19)$$

$$w_t = (1 - m_{t-1}) \cdot \frac{\kappa_{t,t} - \kappa_{t-1,t-1} \cdot \frac{(1 - m_t)^2}{(1 - m_{t-1})^2}}{\kappa_{t,t}}. \quad (20)$$

545 This setup thus provides training examples of localized flows which can be used to learn a parameterized target-agnostic transformation that maps from a source  $\mathbf{x}_T$  to a plausible target  $\mathbf{x}_0$  without  
546 knowing it beforehand using the loss in Eq. 8. To implement this loss, we parameterize the transition  
547 probability of the aforementioned target-agnostic transformation  $p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_T)$  as:  
548

$$p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_T) = \mathbb{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_{\theta}(\mathbf{x}_t, \mathbf{x}_T, t), \tilde{\kappa}_{t-1} \cdot \mathbf{I}), \quad (21)$$

549 where the  $\boldsymbol{\mu}_{\theta}$  is parameterized following Eq. 16:

$$\boldsymbol{\mu}_{\theta}(\mathbf{x}_t, \mathbf{x}_T, t) = u_t \cdot \mathbf{x}_t + v_t \cdot \mathbf{x}_T + w_t \cdot \epsilon_{\theta}(\mathbf{x}_t, t), \quad (22)$$

550 which features a parameterized noise prediction network  $\epsilon(\mathbf{x}_t, t)$  to predict the noise perturbed  
551 quantity  $m_t(\mathbf{x}_T - \mathbf{x}_0) + \sqrt{\kappa_{t,t}} \epsilon$  of a local flow at time  $t$ . Under this parameterization, the training  
552 loss in Eq. 8 can be simplified as,

$$\boldsymbol{\theta}_{\text{PB}} = \arg \min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_0, \mathbf{x}_T, \epsilon} \left[ \|m_t(\mathbf{x}_T - \mathbf{x}_0) + \sqrt{\kappa_{t,t}} \epsilon - \epsilon_{\theta}(\mathbf{x}_t, t)\|^2 \right]. \quad (23)$$

It is then approximately optimized via sampling  $(\mathbf{x}_T, \mathbf{x}_0)$  from the synthetic dataset  $D_s$  created using Algorithm 1 above. For a more practical implementation, we further leverage the corresponding scores  $y_T$  and  $y_0$  of  $\mathbf{x}_T$  and  $\mathbf{x}_0$  which are also included in  $D_s$ . In particular, we further parameterize  $\epsilon_\theta(\mathbf{x}_t, t)$  as  $\epsilon_\theta(\mathbf{x}_t, t, (1 - \gamma)y + \gamma\emptyset)$  with  $y = (y_0, y_T)$  and  $\gamma \sim \text{Ber}(\rho)$  following similar practice in guided diffusion [22]. The intuition is that we want to leverage the output scores to guide the noise prediction network but at the same time, we do not want the network to overly depend on such guidance. This is enforced using the dropout trick which randomly removes the guiding information during training with probability  $\rho \in (0, 1)$ . The above learning loss can then be recast as:

$$\theta_{\text{PB}}^* \triangleq \arg \min_{\theta} \mathbb{E}_{\mathbf{x}_0, y_0, \mathbf{x}_T, y_T, \epsilon} \left[ \|m_t(\mathbf{x}_T - \mathbf{x}_0) + \sqrt{\kappa_{t,T}} \epsilon - \epsilon_\theta(\mathbf{x}_t, t, (1 - \gamma) \cdot y + \gamma \cdot \emptyset)\|^2 \right], \quad (24)$$

where  $\gamma \sim \text{Ber}(\rho)$ . Once trained, the noise network  $\epsilon_\theta$  is used as below during the simulation phase:

$$\epsilon_\theta(\mathbf{x}_t, t, y) = (1 + \beta) \cdot \epsilon_\theta(\mathbf{x}_t, t, y) - \beta \cdot \epsilon_\theta(\mathbf{x}_t, t, \emptyset), \quad (25)$$

where  $\beta$  is the classifier free guidance weight and  $y = [y_T, \alpha \cdot y_*]$  with  $y_*$  denotes the maximum oracle score. Our detail algorithm is presented in Algorithm 2.

For more details, we utilize an MLP  $\epsilon_\theta(\mathbf{x}_t, t, y)$  comprising four layers, each with 1024 units. Each layer employs the Swish activation function,  $\text{Swish}(z) = z\sigma(z)$ , where  $\sigma(z)$  denotes the sigmoid function. The MLP is trained using the Adam optimizer for 100 epochs with a learning rate of 0.001. During each epoch, we sample  $n_e = 8$  synthetic functions from the Gaussian process (the total number of synthetic functions is  $n_g = n_e \times E = 8 \times 100 = 800$  functions) and generate  $n_p = 1024$  samples for each function. At the testing phase, we sample high-value design candidates from the 128 best designs in the offline dataset, using  $T = 200$  sequential denoising steps. All hyperparameters for modeling, training, and sampling with our model are summarized in Table 7.

Table 7: Hyperparameters for the generalized Brownian Bridge diffusion process in **ROOT**.

	Hyperparameter	Value
Architecture	Hidden size	1024
	Number of layers	4
	Activation	Swish
Training	Number of epochs ( $E$ )	100
	Number of functions ( $n_e$ ) (per epoch)	8
	Number of data points ( $n_p$ ) (per function)	1024
	Learning rate	0.001
	Optimizer	Adam
	Batch size	64
	Conditional dropout ( $\rho$ )	0.15
Sampling	$\alpha$	0.8
	$\beta$	-1.5
	Denoising steps	200

For reproducibility, our implementation is made available at the following anonymous link: <https://anonymous.4open.science/r/ROOT>

## C Additional Experiment Results

### C.1 Computation Resource

All our experiments were conducted on a system with the following specifications: Ubuntu 20.04.5, a single NVIDIA A100-SXM4-80GB GPU, and CUDA 10.1. Notably, our method requires less than 6GB of GPU memory per run.

### C.2 Additional Performance Evaluation of ROOT

In the main text, we have reported the 100th percentile scores. In this section, we present additional evaluation results at 80<sup>th</sup> and 50<sup>th</sup> percentiles, providing further insights into the performance distribution of our **ROOT**.

### C.2.1 Performance Evaluation at 80<sup>th</sup> Percentile

As shown in Table 8, our method **ROOT** consistently demonstrates strong performance at the 80<sup>th</sup> percentile level, achieving the best mean rank, i.e., **1.5**. Notably, in the **Ant** and **D’Kitty** tasks, **ROOT** achieves significant improvements over all baselines with remarkable score differences of **0.098** and **0.025** over the runner-ups in those tasks, respectively. **ROOT** also achieves best result in **TF-Bind-8**, outperforming the runner-up with a margin of **0.011**. In the **TF-Bind-10**, **ROOT** performs competitively to the best baselines. Overall, these results demonstrate the consistent reliability and stability of our model’s performance.

Table 8: Experiments on Design-Bench Tasks. We report 80<sup>th</sup> percentile score among  $Q = 128$  candidates. **Blue** denotes the best entry in the column, and **Brown** denotes the second best. **Mean Rank** means the average rank of the method over all the experiment benchmarks.

Method	Benchmarks				Mean Rank
	Ant	D’Kitty	TFBind8	TFBind10	
$D_o$ (best)	0.565	0.884	0.439	0.467	-
BO-qEI	0.629 $\pm$ 0.000	0.884 $\pm$ 0.000	0.439 $\pm$ 0.000	0.510 $\pm$ 0.011	11.00 / 15
CMA-ES	0.007 $\pm$ 0.013	0.718 $\pm$ 0.001	0.652 $\pm$ 0.017	<b>0.543 <math>\pm</math> 0.013</b>	9.50 / 15
REINFORCE	0.182 $\pm$ 0.017	0.562 $\pm$ 0.197	0.622 $\pm$ 0.030	0.519 $\pm$ 0.007	11.25 / 15
GA	0.189 $\pm$ 0.014	0.762 $\pm$ 0.036	<b>0.828 <math>\pm</math> 0.027</b>	0.516 $\pm$ 0.004	8.75 / 15
COMs	0.635 $\pm$ 0.031	0.887 $\pm$ 0.004	0.738 $\pm$ 0.027	0.526 $\pm$ 0.012	5.25 / 15
CbAS	0.542 $\pm$ 0.034	0.813 $\pm$ 0.012	0.585 $\pm$ 0.030	0.517 $\pm$ 0.008	10.25 / 15
MINs	0.746 $\pm$ 0.011	0.908 $\pm$ 0.004	0.545 $\pm$ 0.031	0.519 $\pm$ 0.010	6.50 / 15
RoMA	0.298 $\pm$ 0.033	0.738 $\pm$ 0.018	0.661 $\pm$ 0.010	0.525 $\pm$ 0.003	9.00 / 15
DDOM	<b>0.749 <math>\pm</math> 0.029</b>	0.865 $\pm$ 0.009	0.526 $\pm$ 0.017	0.506 $\pm$ 0.004	9.75 / 15
ICT	0.708 $\pm$ 0.019	0.898 $\pm$ 0.004	0.667 $\pm$ 0.035	0.525 $\pm$ 0.016	6.00 / 15
Tri-mentoring	0.722 $\pm$ 0.015	0.902 $\pm$ 0.003	0.683 $\pm$ 0.047	<b>0.531 <math>\pm</math> 0.007</b>	<b>4.00 / 15</b>
GTG	0.725 $\pm$ 0.077	<b>0.913 <math>\pm</math> 0.007</b>	0.611 $\pm$ 0.038	0.506 $\pm$ 0.006	7.75 / 15
LTR	0.679 $\pm$ 0.016	0.902 $\pm$ 0.002	0.734 $\pm$ 0.025	0.508 $\pm$ 0.010	7.00 / 15
GABO	0.016 $\pm$ 0.009	0.705 $\pm$ 0.002	0.676 $\pm$ 0.021	0.512 $\pm$ 0.008	11.25 / 15
<b>ROOT (ours)</b>	<b>0.847 <math>\pm</math> 0.005</b>	<b>0.938 <math>\pm</math> 0.002</b>	<b>0.839 <math>\pm</math> 0.015</b>	0.526 $\pm$ 0.007	<b>1.50 / 15</b>

### C.2.2 Performance Evaluation at 50<sup>th</sup> Percentile

Table 9 reports the results achieved by all baselines at 50-th percentile. Consistent to or earlier observations, **ROOT** again achieves the best mean rank of **2.0**. In particular, we surpass the runner-up baseline, Gradient Ascent (GA), in the **TF-Bind-8** task with a significant score difference of **0.072**, securing the top rank among all other methods. In the **Ant** and **D’Kitty** tasks, **ROOT** also achieves substantial score gaps of **0.067** and **0.018** over the corresponding runner-ups, respectively. Furthermore, the reported standard deviations of **ROOT** in those tasks are **0.014** and **0.003** which are much lower than those of other baselines. This further ascertains the stability and superior performance of our method across the evaluation benchmark.

### C.3 Score Distribution of ROOT

To provide a more holistic comparison between the score distribution of **ROOT** and those of others, we collect the design candidates obtained from 8 runs ( $128 \times 8 = 1024$  designs) to plot the distribution of scores achieved by **ROOT** and several other representative baselines. The results are shown in Fig. 2. Due to the wide range of scores produced by CMA-ES on the **Ant** task, we separately plot and compare its score distribution against **ROOT**. The remaining plots visualize and compare the score distributions of **ROOT** and 5 representative baselines. In each plot, we annotate the max and median lines to ease the visual comparison. In particular, Fig. 2 reveals that **ROOT** often has a score distribution skewed towards higher-value regions, especially in the **D’Kitty** task. In the **Ant** task, although **ROOT**’s max score is not as high as that of CMA-ES, CMA-ES only produces a single good design while the rest (in its solution set) perform poorly, as also observed in Table 8

Table 9: Experiments on Design-Bench Tasks. We report 50<sup>th</sup> percentile score among  $Q = 128$  candidates. **Blue** denotes the best entry in the column, and **Brown** denotes the second best. **Mean Rank** means the average rank of the method over all the experiment benchmarks.

Method	Benchmarks				Mean Rank
	Ant	D’Kitty	TFBind8	TFBind10	
$D_o$ (best)	0.565	0.884	0.439	0.467	-
BO-qEI	$0.569 \pm 0.000$	$0.883 \pm 0.000$	$0.439 \pm 0.000$	$0.469 \pm 0.005$	7.75 / 15
CMA-ES	$-0.043 \pm 0.007$	$0.674 \pm 0.016$	$0.536 \pm 0.012$	<b><math>0.490 \pm 0.015</math></b>	8.75 / 15
REINFORCE	$0.140 \pm 0.026$	$0.510 \pm 0.203$	$0.450 \pm 0.024$	$0.470 \pm 0.010$	11.00 / 15
GA	$0.137 \pm 0.014$	$0.591 \pm 0.132$	<b><math>0.603 \pm 0.045</math></b>	$0.469 \pm 0.006$	8.75 / 15
COMs	$0.471 \pm 0.034$	$0.862 \pm 0.003$	$0.598 \pm 0.031$	$0.475 \pm 0.010$	5.75 / 15
CbAS	$0.369 \pm 0.008$	$0.748 \pm 0.016$	$0.441 \pm 0.021$	$0.465 \pm 0.006$	10.75 / 15
MINs	$0.618 \pm 0.016$	$0.889 \pm 0.003$	$0.421 \pm 0.017$	$0.467 \pm 0.010$	7.25 / 15
RoMA	$0.224 \pm 0.020$	$0.545 \pm 0.170$	$0.519 \pm 0.073$	<b><math>0.518 \pm 0.003</math></b>	8.50 / 15
DDOM	$0.568 \pm 0.066$	$0.814 \pm 0.016$	$0.404 \pm 0.012$	$0.456 \pm 0.002$	11.00 / 15
ICT	$0.554 \pm 0.018$	$0.872 \pm 0.007$	$0.557 \pm 0.031$	$0.457 \pm 0.033$	8.25 / 15
Tri-mentoring	$0.572 \pm 0.016$	$0.884 \pm 0.001$	$0.562 \pm 0.051$	$0.475 \pm 0.009$	<b><math>4.25 / 15</math></b>
GTG	<b><math>0.645 \pm 0.098</math></b>	<b><math>0.901 \pm 0.005</math></b>	$0.460 \pm 0.032$	$0.452 \pm 0.010$	7.25 / 15
LTR	$0.568 \pm 0.016$	$0.885 \pm 0.002$	$0.566 \pm 0.026$	$0.466 \pm 0.011$	6.00 / 15
GABO	$-0.039 \pm 0.004$	$0.674 \pm 0.005$	$0.496 \pm 0.011$	$0.457 \pm 0.008$	11.50 / 15
<b>ROOT (ours)</b>	<b><math>0.712 \pm 0.014</math></b>	<b><math>0.919 \pm 0.003</math></b>	<b><math>0.675 \pm 0.026</math></b>	$0.473 \pm 0.004$	<b><math>2.00 / 15</math></b>

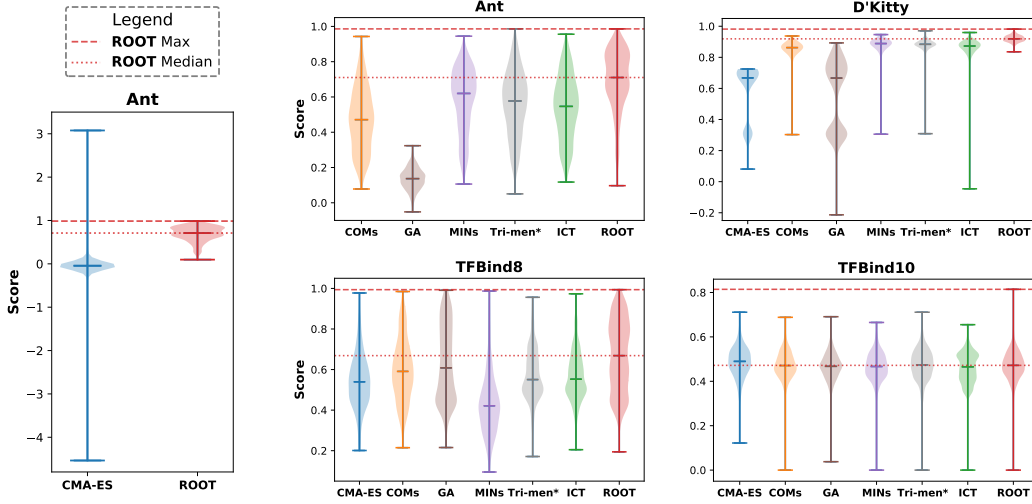


Figure 2: Score distribution of found candidates of **ROOT** compared to others. To avoid cluttering the plots with long chunks of text, we use the abbreviation **Tri-men\*** to denote **Tri-mentoring**.

and Table 9. Otherwise, it can be observed consistently across all plots **ROOT** achieves better score distributions compared to other baselines.

**Summary.** In conclusion, all reported results have demonstrated consistently that our proposed method **ROOT** is, on average, the best-performing baseline in both score distributions and targeted evaluation percentiles. This consistent performance not only highlights the effectiveness and stability of **ROOT** but also reaffirms the robustness of our approach across a wide-range of tasks.

#### C.4 Computational Complexity

**Time Complexity.** The overall computational complexity of our method is determined by three primary components: Gaussian process (GP) fitting, synthetic data generation, and training the

probabilistic bridge model. Given  $n$  offline data points, fitting a GP incurs a computational cost of  $O(n^3)$  due to the inversion of the kernel matrix. For synthetic data generation, we sample  $n_e$  functions (per step) and perform  $M$  gradient steps to generate  $n_p$  synthetic points per function. Each gradient step involves querying the GP, resulting in a total cost of  $O(Mn_en_p n)$ . The probabilistic bridge training component requires  $T$  diffusion steps per sample, where each step involves a forward and backward pass through the score network  $\epsilon_\theta$ , giving a total cost of  $O(T|\theta|n)$ , where  $|\theta|$  denotes the cost of evaluating  $\epsilon_\theta$ . Combining these components, the overall computational complexity is:

$$O(n^3 + n_e M n_p n + T|\theta|n) \approx O(n^3),$$

We note that the time complexity can be further reduced by sub-sampling the offline dataset when fitting the GP, rather than using the entire offline dataset. This approach is analogous to traditional data sampling strategies used in large-scale learning.

**Empirical Runtime.** We empirically evaluate the runtime of our method and compare it against several baselines across a range of tasks. As shown in Table 10, the results demonstrate that **ROOT** achieves faster execution time than several widely used baselines, including CMA-ES, MINS, and Tri-mentoring, while maintaining competitive or superior performance. All experiments are conducted on a **single GPU**, with each task requiring approximately **4GB** of memory.

Table 10: Runtime (in seconds) of different models across four tasks.

Model	Ant	D’Kitty	TF-Bind-8	TF-Bind-10
GA	122	200	124	420
BO-QEI	219	310	402	383
CMA-ES	5747	4889	2667	4323
MINS	797	998	2213	792
REINFORCE	240	257	507	373
CBAS	394	390	835	529
ICT	188	189	216	639
Tri-mentoring	4276	3921	4673	3410
<b>ROOT</b>	298	297	407	575

634

## 635 C.5 Effectiveness of Gaussian process for Generating Synthetic Data.

### 636 C.5.1 DNN-based Approach for Generating Synthetic Data

There are several strategies for sampling closed-form functions fitted to offline data (to provide functions which are similar to the oracle). In **ROOT**, we adopt a standard Gaussian process (GP) approach due to its efficiency, ease of implementation, and strong empirical performance. GPs are particularly advantageous for their simplicity in debugging and their effectiveness in modeling predictive uncertainty. Our motivation is further motivated by ExPT [31], a recent few-shot offline optimization baseline, which similarly employs GPs to generate pseudo labels for offline data.

While alternative generative models exist, they often introduce additional complexity and computational overhead. For instance, we conducted experiments using deep neural networks (DNNs) in place of GPs. One such approach involves randomly initializing the weights of a DNN and training it to fit the offline dataset, then serving as a single sampled function. However, this approach is computationally expensive. Generating 800 such functions using DNNs, each requiring 800 separate training runs, takes approximately **100** minutes, compared to under **300** seconds for the entire training process using our GP-based method.

To further investigate this, we evaluated two less costly alternative setups: (1) an ensemble of five DNNs replacing all 800 GP functions, denoted as **DNN5**, and (2) a single DNN replacing all 800 GP sampled functions, denoted as **DNN1**. As shown in the Table 11, both alternatives perform worse than our GP-based approach and incur significantly higher computational costs.

### 654 C.5.2 Bins-based Approach for Generating Synthetic Data

Beyond the DNN-based approach, we also explore two other heuristic methods that do not depend on Gaussian processes (GP). The first method, *2 bins*, partitions the offline data into two bins, the lowest

Table 11: Performance and runtime of different methods for sampling function on the **Ant** task.

Model	Performance	Time (s)
DNN5	$0.637 \pm 0.268$	1297
DNN1	$0.630 \pm 0.201$	751
<b>ROOT (Ours)</b>	<b><math>0.965 \pm 0.014</math></b>	<b>298</b>

50th percentile and the highest 50th percentile, and sample low- and high-value designs ( $\mathbf{X}^-$ ,  $\mathbf{X}^+$ ) from the corresponding bins respectively. The second method, *64 bins*, divides the data into 64 bins based on the  $\mathbf{y}$  values and then sample  $\mathbf{X}^-$  from the lowest bin and  $\mathbf{X}^+$  from the highest. This approach is similar to the sampling strategy in [26], where trajectories with increasing outputs are selected from offline data to train a model that progressively guides designs from the lowest to the highest bin. In addition, we also examine another GP-based approach that utilizes only one GP function to generate synthetic data. As reported in Table 12, the GP-based methods consistently outperform the above bin-based heuristic approaches, with GP (800 functions) achieving the best performance.

Table 12: Effectiveness of Gaussian process for generating synthetic data.

	Type	Ant	D’Kitty	TF-Bind-8	TF-Bind-10
No-GP	2 bins	$0.745 \pm 0.097$	$0.952 \pm 0.007$	$0.775 \pm 0.057$	$0.641 \pm 0.034$
	64 bins	$0.941 \pm 0.020$	$0.952 \pm 0.009$	$0.837 \pm 0.055$	$0.651 \pm 0.054$
GP	GP (1 function)	$0.955 \pm 0.013$	$0.971 \pm 0.004$	$0.984 \pm 0.012$	$0.657 \pm 0.029$
	GP (800 functions)	<b><math>0.965 \pm 0.014</math></b>	<b><math>0.972 \pm 0.005</math></b>	<b><math>0.986 \pm 0.007</math></b>	<b><math>0.685 \pm 0.053</math></b>

## C.6 GP Hyperparameters Sampling Methods

To diversify the GP mean functions, various approaches can be used to sample GP hyperparameters. In our work, we adopt a simple strategy by uniformly sampling the GP hyperparameters, i.e., the lengthscale  $\ell_s$  and variance  $\sigma_s$ , from a fixed range, as presented in line 5 of Algorithm 1. This approach follows the prior practice in ExPT [31], which also samples these hyperparameters uniformly from a small range and achieves strong performance.

To explore alternative strategies, we have conducted additional experiments. In a **MAP-based** setting, we first optimize the hyperparameters  $\ell_0$  and  $\sigma_0$  by minimizing the negative log marginal likelihood (NLML) on  $D_o = \{\mathbf{x}_i, y_i\}_{i=1}^n$ , which is commonly used in Gaussian processes:

$$\mathcal{L}_{\text{NLML}}(\phi) = \frac{1}{2} \mathbf{y}^\top K_\phi^{-1} \mathbf{y} + \frac{1}{2} \log |K_\phi| + \frac{n}{2} \log 2\pi,$$

where  $K_\phi$  is the kernel matrix parameterized by  $\phi = \{\ell_0, \sigma_0\}$ , and  $n$  is the number of training data points. After obtaining these MAP estimates, we sample new hyperparameters uniformly around them:  $\ell_s \sim [\ell_0 - \delta, \ell_0 + \delta]$  and  $\sigma_s \sim [\sigma_0 - \delta, \sigma_0 + \delta]$ , instead of setting  $\ell_0, \sigma_0$  as in Table 6.

In addition, we have also experimented with an **MCMC-based** approach to sample these hyperparameters. In this setting, we first define a prior distribution  $p(\ell_s, \sigma_s)$  (e.g., a Gaussian prior) and a likelihood  $p(D_o | \ell_s, \sigma_s)$ , derived similarly from the marginal likelihood. We then apply MCMC sampling to draw  $n_g$  hyperparameter combinations from the posterior distribution, i.e.  $p(\ell_s, \sigma_s | D_o) \propto p(D_o | \ell_s, \sigma_s) \times p(\ell_s, \sigma_s)$ .

As shown in Table 13, our uniform sampling method consistently outperforms the aforementioned alternatives. Also, both alternatives introduce significant computational overhead. In particular, the MCMC-based approach requires substantial runtime, incurring approximately **1000** seconds per run, even when using only 8 warmup steps. We leave further exploration of more sophisticated hyperparameter sampling strategies to future work.

## C.7 Hyper-parameter Tuning

**Number of Gradient Steps  $M$ .** We experimented with various numbers of gradient steps ( $M$ ), from the set  $\{25, 50, 75, 100\}$  to construct  $\mathbf{X}_s^-$  and  $\mathbf{X}_s^+$  during the data generation phase (see



Table 13: Performance and runtime of different GP hyperparameter sampling methods on **Ant** task.

Method	Performance
MAP ( $n_g = 800$ )	$0.940 \pm 0.023$
MCMC ( $n_g = 200$ )	$0.884 \pm 0.017$
MCMC ( $n_g = 400$ )	$0.921 \pm 0.010$
MCMC ( $n_g = 600$ )	$0.932 \pm 0.016$
MCMC ( $n_g = 800$ )	$0.916 \pm 0.009$
<b>ROOT (<math>n_g = 800</math>)</b>	<b><math>0.965 \pm 0.014</math></b>

691 Section 3.3). Our experiments reveal that increasing  $M$  consistently improves the overall performance  
692 of the algorithm as illustrated in Table 14. Increasing the number of gradient steps allows our  
693 model to more precisely distinguish between low-value and high-value regions in the distribution,  
694 substantially enhancing our performance. However, increasing the no. of gradient steps also increases  
695 computational time, so we select  $M = 100$  as the best balance between performance and efficiency.

Table 14: Impact of gradient steps  $M$  on **ROOT**.

Steps ( $M$ )	Ant	D’Kitty	TF-Bind-8	TF-Bind-10
25	$0.968 \pm 0.009$	$0.972 \pm 0.003$	$0.952 \pm 0.024$	$0.640 \pm 0.039$
50	$0.968 \pm 0.015$	$0.969 \pm 0.003$	$0.945 \pm 0.025$	$0.639 \pm 0.024$
75	$0.950 \pm 0.011$	$0.969 \pm 0.005$	$0.972 \pm 0.013$	$0.652 \pm 0.033$
100	$0.965 \pm 0.014$	$0.972 \pm 0.005$	$0.986 \pm 0.007$	$0.685 \pm 0.053$

Table 15: Impact of number of paired data points on **ROOT**.

Initial Points ( $n_p$ )	Ant	TF-Bind-8
128	$0.915 \pm 0.020$	$0.948 \pm 0.024$
256	$0.950 \pm 0.010$	$0.968 \pm 0.018$
512	$0.964 \pm 0.010$	$0.974 \pm 0.006$
1024	$0.965 \pm 0.014$	$0.986 \pm 0.007$

696 **Number of Initial Points.** For each synthetic function generated by the Gaussian process, we  
697 will draw a number of initial data points from the offline dataset to initiate the exploration into the  
698 low-value and high-value regions via gradient descent and ascent, respectively. We experimented  
699 with different numbers of initial points from the set  $\{128, 256, 512, 1024\}$ . As shown in the Table 15,  
700 this consistently led to improved performance. This observation is similar to a previous observation  
701 that having more well-curated training data tends to enhance the overall model performance. We  
702 selected 1024 as the best balance between computational cost and performance score.

703 **Number of GP mean functions.** We additionally conducted experiments on the number of GP mean  
704 functions to generate synthetic data. In our original method, we use  $n_g = n_e \times M = 800$  functions;  
705 here, we vary this number from 100 to 1000. The empirical results in Table 16 indicate that the final  
706 performance improves as the number of Gaussian Processes (GPs) increases. However, there may  
707 exist a saturation point (e.g., 800 GPs), beyond which further increasing the number of GPs could  
708 lead to a decline in performance.

Table 16: Performance on **Ant** and **TF-Bind-10** with varying number of GPs.

#GPs	Ant	TF-Bind-10
100	$0.914 \pm 0.015$	$0.632 \pm 0.020$
200	$0.914 \pm 0.015$	$0.677 \pm 0.038$
400	$0.959 \pm 0.010$	$0.631 \pm 0.018$
600	$0.956 \pm 0.021$	$0.674 \pm 0.044$
<b>800 (ROOT)</b>	<b><math>0.965 \pm 0.014</math></b>	<b><math>0.685 \pm 0.053</math></b>
1000	$0.960 \pm 0.016$	$0.662 \pm 0.033$

709 **GP hyperparameters.** We also employed an ablation study to see the effect of Gaussian hyperpa-  
710 rameters (i.e., initial lengthscale  $\ell_0$ ) on our final results. As can be seen clearly from the Fig. 3, the  
711 optimal range for  $\ell_0$  yields around 1.0 for the continuous tasks and 6.25 for the discrete tasks.

712 **GP hyperparameters range size  $\delta$  and step size  $\eta$ .** In the data-generation phase, we uniformly  
713 sample the Gaussian process lengthscale and variance from the intervals  $[\ell_0 - \delta, \ell_0 + \delta]$  and  
714  $[\sigma_0 - \delta, \sigma_0 + \delta]$ , respectively (see Line 5 Algorithm 1). Choosing  $\delta$  is a trade-off: if it is too large,  
715 the probabilistic bridge will learn on functions that deviate excessively from the oracle, but if it is too  
716 small, we lose the diversity needed for robust generalization. We conducted a sweep over different

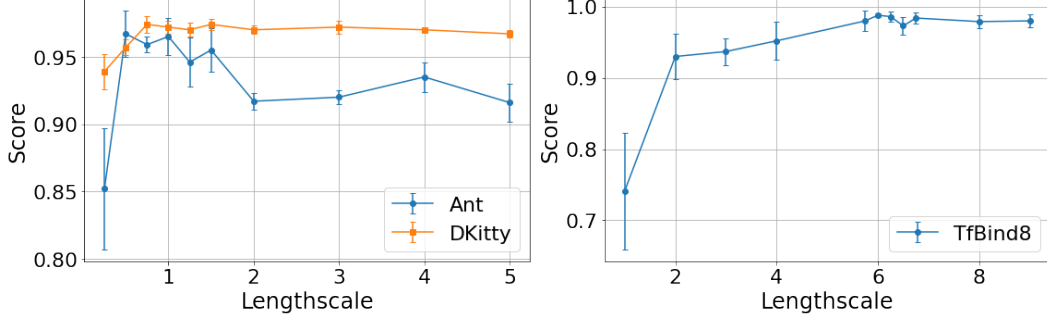


Figure 3: Effect of lengthscale on performance for Ant and DKitty.

values of  $\delta$ ; as shown in Table 17,  $\delta = 0.25$  achieves the best performance across both the **Ant** and **TF-Bind-8** tasks. In addition, during this phase we apply  $M$  gradient steps with fixed step size  $\eta$  (see Eq.12 and Eq. 13). Our experiments in Table 18 demonstrate that for continuous tasks, setting  $\eta = 0.001$  achieves the best score.

Table 17: Performance on **Ant** and **TF-Bind-8** with varying  $\delta$ .

$\delta$	<b>Ant</b>	<b>TF-Bind-8</b>
0.05	$0.953 \pm 0.011$	$0.982 \pm 0.005$
0.10	$0.957 \pm 0.014$	$0.986 \pm 0.006$
<b>0.25 (ours)</b>	$0.965 \pm 0.014$	$0.986 \pm 0.007$
0.50	$0.959 \pm 0.013$	$0.981 \pm 0.007$
1.00	$0.959 \pm 0.014$	$0.989 \pm 0.003$

Table 18: Performance on **Ant** and **D’Kitty** with varying  $\eta$ .

$\eta$	<b>Ant</b>	<b>D’Kitty</b>
0.0005	$0.969 \pm 0.017$	$0.971 \pm 0.003$
0.00075	$0.968 \pm 0.013$	$0.969 \pm 0.005$
<b>0.001 (ours)</b>	$0.965 \pm 0.014$	$0.972 \pm 0.005$
0.0025	$0.964 \pm 0.010$	$0.976 \pm 0.004$
0.005	$0.948 \pm 0.011$	$0.977 \pm 0.003$

## C.8 Score and Pseudo-Value Distribution of Generated Samples from GP

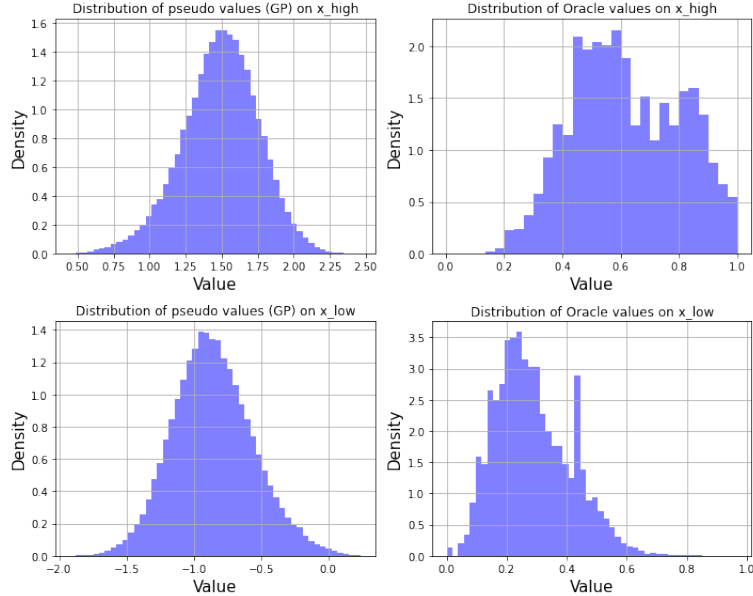


Figure 4: Distribution of pseudo-values (GP) and oracle values on low- and high-value regions.

To further demonstrate the efficiency of Gaussian processes, we conducted an experiment on the **Ant** task using low and high regions produced by 800 GP mean functions. Each function produces 1024

low-value points ( $\mathbf{X}^-$  or  $\mathbf{x}_{\text{low}}$ ) along with 1024 high-value points ( $\mathbf{X}^+$  or  $\mathbf{x}_{\text{high}}$ ). We concatenate all  $1024 \times 800$  points to form the distribution for the low-value and high-value designs. We then plotted both the oracle value distribution by using the oracle function to evaluate and the pseudo-value distribution ( $y^-$  and  $y^+$  predicted by GP). As shown in Figure 4, the low and high regions correspond to two clearly separated distributions. The oracle values in  $\mathbf{X}^+$  are skewed toward higher values, while those in  $\mathbf{X}^-$  are skewed toward lower values. This separation facilitates effective training of our probabilistic bridge model as it learns to transform between two distinct distributions.

### C.9 Extreme Limited Budget Settings

While a budget of  $Q = 128$  candidates is a commonly adopted setting, we also conduct experiments under more constrained budgets to evaluate the robustness of our method. The results, shown in Table 19, demonstrate that **ROOT** continues to achieve strong performance even in these highly limited-query scenarios, outperforming existing baselines.

Table 19: Performance of different methods under extreme limited offline data budgets ( $Q$ ). **ROOT** consistently outperforms baselines on both **Ant** and **TF-Bind-8**.

Q Budget	Method	Ant	TFBind8
16	GA	$0.250 \pm 0.033$	$0.938 \pm 0.020$
	COMs	$0.789 \pm 0.070$	$0.917 \pm 0.059$
	<b>ROOT</b>	$0.938 \pm 0.022$	$0.956 \pm 0.028$
32	GA	$0.252 \pm 0.036$	$0.961 \pm 0.024$
	COMs	$0.776 \pm 0.033$	$0.871 \pm 0.030$
	<b>ROOT</b>	$0.942 \pm 0.019$	$0.974 \pm 0.020$
64	GA	$0.281 \pm 0.023$	$0.978 \pm 0.020$
	COMs	$0.834 \pm 0.051$	$0.922 \pm 0.036$
	<b>ROOT</b>	$0.942 \pm 0.020$	$0.974 \pm 0.021$

## D Broader Impact and Limitation

**Broader Impact.** This work provides a novel lens for offline black-box optimization by reframing it as a distributional translation problem, potentially inspiring new probabilistic modeling techniques in low-data regimes. The approach opens up new possibilities for data-efficient optimization in applications where function evaluations are expensive or infeasible. These include (but are not limited to) materials discovery, policy design, and automated experimentation. However, caution must be exercised when deploying such methods in safety-critical domains, as reliance on synthetic priors may introduce epistemic uncertainty that requires careful quantification and calibration.

**Limitation.** One practical consideration is the additional computational overhead introduced by fitting multiple Gaussian processes to construct the synthetic function ensemble. While this step is performed offline and enables better data efficiency during optimization, it may require tuning and parallelization to scale effectively with large input dimensions or limited compute resources.

## E Other Practical Setups for Our Probabilistic Bridge Framework

In this paper, we employ the Brownian bridge as our practical probabilistic bridge (see Appendix B.2), which achieves significant performance across our experiments. However, our proposal method is a flexible framework that can be universally adapted to other probabilistic bridges. In this section, we will demonstrate another design: **Ornstein-Uhlenbeck Bridge**.

From our definition of the probabilistic bridge in Section 3.1.1, we can choose  $\psi_t$  and  $\kappa_{t,k}$  as

$$\psi_t(\mathbf{x}_0, \mathbf{x}_T) = \mathbf{x}_0 \cdot \frac{\sinh(\alpha(T-t))}{\sinh(\alpha T)} + \mathbf{x}_T \cdot \frac{\sinh(\alpha t)}{\sinh(\alpha T)}, \quad (26)$$

$$\kappa_{t,k} = \frac{\sinh(\alpha \cdot \min(t, k)) \cdot \sinh(\alpha \cdot (T - \max(t, k)))}{\alpha \cdot \sinh(\alpha T)}. \quad (27)$$

Table 20: Performance of **ROOT** with different bridge configuration across the benchmark tasks.

Bridges	Ant	D’Kitty	TF-Bind-8	TF-Bind-10
<b>ROOT</b> (BBDM)	0.965 ± 0.014	<b>0.972 ± 0.005</b>	<b>0.986 ± 0.007</b>	<b>0.685 ± 0.053</b>
<b>ROOT</b> (OUDM)	<b>1.940 ± 0.599</b>	0.723 ± 0.001	0.927 ± 0.049	0.675 ± 0.124

This choice will result in the formula of the Ornstein-Uhlenbeck bridge <sup>4</sup> with the hyperparameter  $\alpha$ . Once this bridge is learned, we can simulate  $\mathbf{x}_t$  following the previously derived simulation approach in Eq. 5 which, under the OU instantiation, becomes,

$$\mathbf{x}_t = \mathbf{x}_0 \cdot \frac{\sinh(\alpha(T-t))}{\sinh(\alpha T)} + \mathbf{x}_T \cdot \frac{\sinh(\alpha t)}{\sinh(\alpha T)} + \sqrt{\frac{\sinh(\alpha t) \cdot \sinh(\alpha(T-t))}{\alpha \cdot \sinh(\alpha T)}} \cdot \boldsymbol{\epsilon}_t \quad (28)$$

where  $\boldsymbol{\epsilon}_t \sim \mathbb{N}(0, \mathbf{I})$ . From the above equation, we can also represent  $\mathbf{x}_0$  in terms of  $\mathbf{x}_t$  and  $\mathbf{x}_T$ :

$$\mathbf{x}_0 = \left( \mathbf{x}_t - \mathbf{x}_T \cdot \frac{\sinh(\alpha t)}{\sinh(\alpha T)} - \sqrt{\frac{\sinh(\alpha t) \cdot \sinh(\alpha(T-t))}{\alpha \cdot \sinh(\alpha T)}} \cdot \boldsymbol{\epsilon}_t \right) \cdot \frac{\sinh(\alpha T)}{\sinh(\alpha(T-t))} \quad (29)$$

The transition probability of the localized bridge/flow, i.e.,  $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_T)$  in Eq. 6, can be derived by using the conditional Gaussian rule which results in the following transition mean,

$$\boldsymbol{\mu}(\mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_T) = \psi_{t-1}(\mathbf{x}_0, \mathbf{x}_T) + \kappa_{t-1,t} \kappa_{t,t}^{-1} (\mathbf{x}_t - \psi_t(\mathbf{x}_0, \mathbf{x}_T)) \quad (30)$$

Substituing  $\mathbf{x}_t = \psi_t(\mathbf{x}_0, \mathbf{x}_T) + \sqrt{\kappa_{t,t}} \cdot \boldsymbol{\epsilon}_t$ , the above transition mean can be rewritten as:

$$\boldsymbol{\mu}(\mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_T) = \psi_{t-1}(\mathbf{x}_0, \mathbf{x}_T) + \kappa_{t-1,t} \kappa_{t,t}^{-1/2} \boldsymbol{\epsilon}_t \quad (31)$$

$$= \mathbf{x}_0 \frac{\sinh(\alpha(T-t+1))}{\sinh(\alpha T)} + \mathbf{x}_T \frac{\sinh(\alpha(t-1))}{\sinh(\alpha T)} + \boldsymbol{\epsilon}_t \frac{\sinh(\alpha(t-1)) \sinh(\alpha(T-t))}{\sqrt{\alpha \sinh(\alpha T) \sinh(\alpha t) \sinh(\alpha(T-t))}} \quad (32)$$

By substituting  $\mathbf{x}_0$  from Eq. 29, we get a closed-form formula for  $\boldsymbol{\mu}(\mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_T) = u_t \cdot \mathbf{x}_t + v_t \cdot \mathbf{x}_T + w_t \cdot \boldsymbol{\epsilon}_t$  where the coefficients are computed as

$$u_t = \frac{\sinh(\alpha(T-t+1))}{\sinh(\alpha(T-t))}, \quad v_t = \left[ \frac{\sinh(\alpha(t-1))}{\sinh(\alpha T)} - \frac{\sinh(\alpha(T-t+1)) \sinh(\alpha t)}{\sinh(\alpha T) \sinh(\alpha(T-t))} \right] \quad (33)$$

$$w_t = \sqrt{\frac{\sinh(\alpha(T-t))}{\alpha \sinh(\alpha T) \sinh(\alpha t)}} \cdot \left[ \sinh(\alpha(t-1)) - \sinh(\alpha t) \cdot \frac{\sinh(\alpha(T-t+1))}{\sinh(\alpha(T-t))} \right] \quad (34)$$

Next, following prior practice, we again reparameterize the target-agnostic transformation  $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_T)$  as a neuralized Gaussian with mean  $\boldsymbol{\mu}_\theta(\mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_T) = u_t \cdot \mathbf{x}_t + v_t \cdot \mathbf{x}_T + w_t \cdot \boldsymbol{\epsilon}_t$  and same covariance matrix as the local transition, the (training) ELBO loss in Eq. 8 can be simplified as:

$$\boldsymbol{\theta}_{\text{PB}} = \arg \min_{\boldsymbol{\theta}} \mathbb{E}_{(\mathbf{x}_0, \mathbf{x}_T) \sim D_s, \boldsymbol{\epsilon}_t \sim \mathbb{N}(0, \mathbf{I})} \left\| \epsilon_\theta(\psi_t(\mathbf{x}_0, \mathbf{x}_T) + \kappa_{t,t} \cdot \boldsymbol{\epsilon}_t, \mathbf{x}_T, t) - \boldsymbol{\epsilon}_t \right\|^2 \quad (35)$$

In addition, incorporating the classifier-free guidance techniques leads to:

$$\boldsymbol{\theta}_{\text{PB}} = \arg \min_{\boldsymbol{\theta}} \mathbb{E}_{(\mathbf{x}_0, \mathbf{y}_0, \mathbf{x}_T, \mathbf{y}_T), \boldsymbol{\epsilon}_t} \left\| \epsilon_\theta(\psi_t(\mathbf{x}_0, \mathbf{x}_T) + \kappa_{t,t} \cdot \boldsymbol{\epsilon}_t, t, \gamma \cdot \mathbf{y} + (1-\gamma) \cdot \emptyset) - \boldsymbol{\epsilon}_t \right\|^2 \quad (36)$$

where  $\gamma \sim \text{Ber}(\rho)$ . Once the training process is done, we can employ the trained network  $\epsilon_\theta$  for our optimization-via-simulation process:

$$\mathbf{x}_{t-1} = u_t \cdot \mathbf{x}_t + v_t \cdot \mathbf{x}_T + w_t \cdot \epsilon_\theta(\mathbf{x}_t, t, \mathbf{y}) + \sqrt{\tilde{\kappa}_{t-1}} \cdot \boldsymbol{\epsilon}, \quad (37)$$

<sup>4</sup>Aria Ahari, Larbi Alili, Massimiliano Tamborrino, *Boundary crossing problems and functional transformations for Ornstein-Uhlenbeck processes*, 2024, <https://doi.org/10.48550/arXiv.2210.01658>

771 where  $\tilde{\kappa}_{t-1} = \kappa_{t-1,t-1} - \kappa_{t-1,t} \kappa_{t,t}^{-1} \kappa_{t,t-1}$  and  $\epsilon \sim \mathbb{N}(0, I)$ .

772 The performance of the OU-based **ROOT** is compared against the original Brownian-based **ROOT** in  
773 Table 20 . The results show that the OU-based variant of **ROOT** performs competitively to the original  
774 Brownian variant on **TF-Bind-8** and **TF-Bind-10**. On the **Ant** task, the OU-based variant is better  
775 while on the **D’Kitty** task, the Brownian variant is better. We leave a more thorough investigation  
776 across different variants of **ROOT** with different bridge configuration to future work as such detailed  
777 investigation is beyond the scope of our current work.