Generative Pre-Training for Black-Box Optimization

Jiahui Zhu

Ph.D Candidate at EECS

April 6, 2026

Outline

1. **Problem setup: offline black-box optimization**
2. **Why prior methods are not enough**
3. **BONET: learn optimization trajectories**
4. **Difference among BONET / Diffusion**

Problem Setup

Offline black-box optimization

Given a black-box objective $f : X \to \mathbb{R}$ and a fixed offline dataset

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}, \qquad y_i = f(x_i),$$

learn from *only* this dataset and propose a new design x with large objective value.

- **Black-box:** no analytic form, no gradient, and querying f is expensive.
- **Offline:** during training we cannot adaptively query the true function.
- Goal: propose candidates that can *outperform the best point in the dataset*.

Online vs. Offline BBO

Online BBO

- Query $f(x_t)$, observe result, then choose x_{t+1}.
- Can balance exploration and exploitation adaptively.
- Typical optimizers naturally produce sequential trajectories.

Offline BBO

- Only a static dataset is available.
- No interaction with the environment during training.
- Hardest issue: distribution shift when proposing OOD designs.

Main challenge

How can we learn an optimizer-like behavior when the training data are *not* trajectories, but only independent samples?

Why Prior Methods Can Fail

Forward / surrogate-based methods

Fit a predictive model $\hat{f}(x)$, then optimize x by maximizing $\hat{f}(x)$.

- Problem: $\hat{f}(x)$ may be badly miscalibrated outside the training distribution.
- Result: optimization is pushed toward unrealistic OOD regions.

Inverse / conditional generative methods

Directly learn "generate x conditioned on a large target value y."

- Problem: the inverse map from target value to design is often highly multimodal.
- Especially difficult in high-dimensional design spaces.

BONET: The Main Idea

Key perspective shift

Instead of learning a static mapping $x \mapsto y$ or $y \mapsto x$, BONET learns the **optimization process itself**.

- In online BBO, optimizers generate a sequence of improving proposals.
- BONET asks: can we pretrain a generative model to imitate this *trajectory structure*?
- Then at test time, use the learned sequence model to rollout better candidates.

Pipeline

Phase 1: build pseudo-trajectories from static offline data.
Phase 2: train an autoregressive model on these trajectories.
Phase 3: condition on a prefix and rollout new high-quality designs.

What Exactly Happens in Phase 1?

Input

A static offline dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$.

Output

A trajectory dataset

$$\mathcal{D}_{\text{traj}} = \{\tau^{(k)}\}_{k=1}^{M},$$

where each trajectory has the form

$$\tau = (R_1, x_1, R_2, x_2, \ldots, R_T, x_T).$$

- $x_1, \ldots, x_T$ are arranged to mimic an optimization path.
- R_i is a regret budget that indicates how much improvement remains.

Phase 1: Sorted Trajectories

Heuristic observation

In many online black-box optimizers, the sequence of proposed designs tends to improve over time on average.

So BONET constructs a pseudo-trajectory by:

1. sampling T points from the offline dataset,
2. sorting them by objective value from low to high.

$$f(x_1) \leq f(x_2) \leq \cdots \leq f(x_T).$$

Interpretation

This is not a true historical trajectory; it is a **synthetic optimization-like sequence** whose quality improves from early to late steps.

Why Uniform Sampling Is Not Enough

- In many datasets, low-value samples are abundant, while high-value samples are rare.
- If we sample uniformly, most pseudo-trajectories will end in mediocre regions.
- Then the model cannot learn how a good optimizer eventually reaches high-value designs.

Needed fix

Before sorting, we should **upweight promising bins** so that the trajectory dataset contains more high-quality suffixes.

Phase 1: SORT-SAMPLE Reweighting

Partition the offline data into N_B equal-width bins according to objective values. For each bin B_i, define the score

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

where:

- $|B_i|$ = number of points in bin i,
- y_{b_i} = center of bin i,
- $\hat{y}$ = best objective value observed in the dataset,
- K, τ control smoothing and concentration.

Meaning

Large s_i means this bin should contribute more points to the pseudo-trajectory dataset.

Understanding the Bin Score s_i

Term 1: bin size factor

$$\frac{|B_i|}{|B_i| + K}$$

- favors bins with more samples,
- preserves diversity,
- avoids over-trusting tiny bins.

Term 2: quality factor

$$\exp\left(-\frac{|\hat{y} - y_{b_i}|}{\tau}\right)$$

- favors bins close to the best value,
- upweights high-quality regions,
- makes good suffixes more likely.

Trade-off

BONET does not only chase the top bin; it balances **quality** and **coverage**.

From s_i to Sampling Counts n_i

After computing the bin scores, BONET allocates the number of selected points proportionally:

$$n_i \propto s_i, \qquad \sum_{i=1}^{N_B} n_i = T.$$

Then it samples n_i points uniformly from each bin B_i and finally sorts them by objective value.

Result

Each pseudo-trajectory contains:

- some low-value points at the beginning,
- more medium/high-value points near the end,
- a shape that resembles "exploration $\rightarrow$ exploitation."

Toy Example for SORT-SAMPLE: Computing s_i

Suppose we divide the offline dataset into 3 bins and set

$$\hat{y} = 9, \qquad K = 10, \qquad \tau = 6, \qquad T = 8.$$

Bin	y_{b_i}	$\lvert B_i \rvert$	Meaning
B_1	-6	40	low
B_2	0	50	mid
B_3	6	30	high

For each bin,

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

$$s_1 = \frac{40}{50} \exp\left(-\frac{|9 - (-6)|}{6}\right) = 0.8e^{-15/6} \approx 0.0657,$$

$$s_2 = \frac{50}{60} \exp\left(-\frac{|9 - 0|}{6}\right) \approx 0.8333e^{-1.5} \approx 0.1859,$$

$$s_3 = \frac{30}{40} \exp\left(-\frac{|9 - 6|}{6}\right) = 0.75e^{-0.5} \approx 0.4549.$$

Thus, the high-value bin receives the largest weight.

Toy Example for SORT-SAMPLE: From s_i to n_i

The total score is

$$s_1 + s_2 + s_3 \approx 0.0657 + 0.1859 + 0.4549 = 0.7065.$$

Normalize them into proportions:

$$p_1 = \frac{s_1}{0.7065} \approx 0.093, \qquad p_2 = \frac{s_2}{0.7065} \approx 0.263, \qquad p_3 = \frac{s_3}{0.7065} \approx 0.644.$$

Since the trajectory length is $T = 8$, the expected allocation is

$$8p_1 \approx 0.74, \qquad 8p_2 \approx 2.10, \qquad 8p_3 \approx 5.15.$$

After rounding to integers, one possible allocation is

$$(n_1, n_2, n_3) = (1, 2, 5).$$

Hence we may sample

$$-8 \mid -1, 2 \mid 5, 6, 7, 8, 9$$

from the three bins, and then sort them into the pseudo-trajectory

$$-8 \to -1 \to 2 \to 5 \to 6 \to 7 \to 8 \to 9.$$

Phase 1: Regret Budget

After building a sorted trajectory $x_1, \ldots, x_T$, BONET attaches a regret budget to each step:

$$R_i = \sum_{j=i}^{T} \big(f(x^*) - f(x_j)\big).$$

- R_i measures how much total regret remains from step i onward.
- Larger R_i: more room for exploration / low-quality proposals.
- Smaller R_i: should already be close to high-quality regions.

Final trajectory representation

$$\tau = (R_1, x_1, R_2, x_2, \ldots, R_T, x_T).$$

This is the sequence fed to the autoregressive model in Phase 2.

Why Regret Budget Matters

Sorting alone gives the model a direction: *early points are worse, later points are better.*

Regret budget adds the missing notion of **pace**:

- how aggressively the sequence should move toward high-value regions,
- how much "optimization budget" remains,
- a controllable signal that can be manipulated at test time.

Takeaway

Phase 1 converts static offline data into optimization-like sequences with both **ordering** and **progress information**.

Phase 2: What Is the Goal?

Input to Phase 2

The trajectory dataset from Phase 1:

$$\mathcal{D}_{\text{traj}} = \{\tau^{(k)}\}_{k=1}^{M}, \qquad \tau = (R_1, x_1, R_2, x_2, \ldots, R_T, x_T).$$

Output of Phase 2

Train a sequence model that predicts the **next design point** from the previous trajectory prefix.

- Phase 1 creates the *training stories*.
- Phase 2 turns those stories into a *learned optimizer*.
- The model should understand both **where it is** and **how fast it should improve**.

Phase 2: Sequence Modeling View

BONET models the conditional distribution

$$g_\theta(x_i \mid x_{<i}, R_{\leq i}),$$

where

$$x_{<i} = (x_1, \ldots, x_{i-1}), \qquad R_{\leq i} = (R_1, \ldots, R_i).$$

Interpretation

At step i, the model looks at the history of previous designs and regret budgets, then predicts what the next design x_i should be.

- This is analogous to language modeling: predict the next token from previous tokens.
- Here, the "tokens" are optimization states rather than words.

Teacher Forcing: What Is Data and What Is Prediction?

Ground-truth from data

A trajectory in $\mathcal{D}_{\text{traj}}$ already contains the true sequence

$$(R_1, x_1, R_2, x_2, \ldots, R_T, x_T).$$

So x_i is the **real next point** stored in the dataset.

Prediction from model

Given the prefix

$$(R_1, x_1, \ldots, R_{i-1}, x_{i-1}, R_i),$$

the model outputs

$$\hat{x}_i = f_\theta(x_{<i}, R_{\leq i}).$$

Training comparison

At step i, we compare the model prediction $\hat{x}_i$ with the ground-truth target x_i.

Training Objective: Maximum Likelihood

For a trajectory τ, Phase 2 maximizes the log-likelihood of the true next points:

$$L(\theta; \mathcal{D}_{\text{traj}}) = \mathbb{E}_{\tau \sim \mathcal{D}_{\text{traj}}} \left[\sum_{i=1}^{T} \log g_\theta(x_i \mid x_{<i}, R_{\leq i}) \right].$$

Meaning

The model is rewarded when it assigns high probability to the actual next point appearing in the trajectory.

- maximize likelihood $\Longleftrightarrow$ learn to imitate good optimization trajectories,
- the model does *not* directly optimize $f(x)$,
- instead it learns the **distribution of improvement steps**.

From Maximum Likelihood to Loss Minimization

In implementation, we usually minimize a loss instead of maximizing the objective above:

$$\max_{\theta} L(\theta) \qquad \Longleftrightarrow \qquad \min_{\theta} \mathcal{L}(\theta), \quad \mathcal{L}(\theta) = -L(\theta).$$

Discrete design space

Use cross-entropy / negative log-likelihood:

$$\mathcal{L}_{\mathrm{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \log p_\theta(x_i \mid x_{<i}, R_{\leq i}).$$

Continuous design space

Use MSE as a regression loss:

$$\mathcal{L}_{\mathrm{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \|x_i - \hat{x}_i\|_2^2 .$$

So x_i is the target from data, and $\hat{x}_i$ is the model prediction.

Why Regret Budget Is Crucial in Phase 2

What sorting already provides

Sorting tells the model the **direction** of improvement: early points are worse, later points are better.

What regret budget additionally provides

Regret budget tells the model the **pace** of improvement: how aggressively the next proposal should move toward high-value regions.

- large R_i: there is still plenty of regret left, so the trajectory can be looser,
- small R_i: the model should already be proposing much stronger designs,
- therefore R_i acts like a signal of **which phase of optimization** we are in.

What Phase 2 Actually Learns

Not just "what point has high score"

Phase 2 does not simply learn a surrogate $x \mapsto y$. It learns how an **optimization process evolves over time**.

The model is effectively learning:

- given previous proposals, what kind of proposal usually comes next;
- given the remaining regret budget, how conservative or aggressive the next step should be;
- how to continue a prefix into a plausible improving suffix.

Takeaway

Phase 2 is where BONET turns pseudo-trajectories into a **generative optimizer** that can later be rolled out in Phase 3.

Phase 3: What Is the Goal?

Test-time optimization, not further training

After Phase 2, the model parameters are fixed. BONET now uses the learned sequence model as a **generative optimizer** on a new evaluation instance.

- Construct a trajectory-like prefix from the offline dataset, following Phase 1.
- Feed this prefix to the trained model as the optimization history.
- Autoregressively roll out the remaining steps to generate new candidate designs.
- Output a small set of candidates for **real black-box evaluation**.

Key point

Phase 1 builds pseudo-trajectories, Phase 2 learns how to continue them, and Phase 3 actually **uses** that learned continuation rule to propose new points.

Phase 3: Prefix Conditioning

Step 1: construct a partial trajectory

BONET first constructs a trajectory

$$\tau' = (R_1, x_1, R_2, x_2, \ldots, R_T, x_T)$$

from the offline dataset using the same SORT-SAMPLE pipeline as in Phase 1. Then it deletes the last $T - P$ points and keeps only the first P steps.

$$(R_1, x_1, R_2, x_2, \ldots, R_P, x_P)$$

- P is the **prefix length**.
- It tells us how much optimization history is given to the model before rollout.
- The prefix acts as a plausible starting context rather than asking the model to generate from scratch.

Interpretation

Phase 3: Autoregressive Rollout

After consuming the prefix, BONET rolls out the model for steps

$$t = P + 1, \ldots, T.$$

At each step, it predicts the next design autoregressively:

$$x_t \sim g_\theta(x_t \mid x_{<t}, R_{\leq t}).$$

During rollout, the regret input is fixed to the user-specified evaluation budget:

$$R_t = \hat{R}, \qquad t = P + 1, \ldots, T.$$

- The prefix provides the **history**.
- The fixed $\hat{R}$ provides the **target pace / target quality level**.
- The model then generates a suffix that is consistent with both.

Important

Phase 3 does *not* update model parameters. It is purely an **inference-time rollout** procedure.

What Do P and Q Mean in Phase 3?

P: prefix length

- Number of steps kept from the constructed trajectory.
- These P points are fed to the model before generation starts.
- Larger P means more context, but fewer steps left to roll out.

Q: query budget

- Maximum number of candidate points finally returned by BONET.
- These are the points we will actually evaluate with the true black-box function, which $|\mathbf{X}| \leq Q$.

Since the model can only generate $T - P$ new points, the final output size is

$$|\mathbf{X}| = \min(Q, T - P).$$

Meaning

P controls the **input history length**, while Q controls the **final evaluation budget**.

Why Return the Last $\min(Q, T - P)$ Rolled-out Points?

Not "best" yet

At rollout time, the generated points have *not* yet been evaluated by the true black-box function, so BONET does not know which one is truly best.

Instead, BONET returns the **last** generated points: $\mathbf{X} = \{x_{T-\min(Q,T-P)+1}, \ldots, x_T\}$.

- In Phase 1, each pseudo-trajectory is sorted from low value to high value.
- Therefore, later positions in the sequence correspond to **more promising regions**.
- Phase 2 learns to continue a prefix toward a higher-quality suffix.
- So in Phase 3, the tail of the rolled-out suffix is the most natural place to extract candidates.

Practical interpretation

BONET first proposes up to Q promising candidates, then the true black-box function evaluates them, and we finally select the best one among those evaluated points.

A Simple Example of Phase 3

Suppose

$$T = 20, \qquad P = 5, \qquad Q = 3.$$

1. Construct a trajectory of length 20 and keep only the first 5 points.
2. Feed
$$(R_1, x_1, \ldots, R_5, x_5)$$
to the trained model as the prefix.
3. Roll out the remaining $20 - 5 = 15$ points:
$$x_6, x_7, \ldots, x_{20}.$$
4. Return the last $\min(3, 15) = 3$ generated points:
$$\mathbf{X} = \{x_{18}, x_{19}, x_{20}\}.$$

These 3 points are then evaluated by the real black-box function, and the best evaluated one is selected as the final proposal.

Phase 3: Final Takeaway

What BONET is doing at test time

Phase 3 turns the trained transformer into a **rollout optimizer**: given an optimization prefix, it generates a plausible improving suffix.

- Phase 1 gives BONET optimization-like sequences.
- Phase 2 teaches BONET how such sequences usually continue.
- Phase 3 uses this learned continuation rule to propose new high-quality candidates.

One-sentence summary

BONET does not directly predict the optimum in one shot; it **continues an optimization trajectory** and outputs a small set of promising candidates for real evaluation.

BONET vs. Guided Diffusion: Core Difference

Summary

BONET learns how an optimizer moves, whereas guided diffusion learns what **high-value solutions look like**.

BONET

- Models an **optimization trajectory**.
- Input is a **prefix** of previous proposals together with the regret budget.
- Predicts **what proposal should come next**.
- Test-time behavior is a **rollout**: keep extending the sequence step by step.

Guided Diffusion

- Models the **distribution of good designs**.
- Starts from noise and gradually denoises toward samples with high target value.
- Uses guidance to bias sampling toward **higher-score regions**.
- Test-time behavior is **conditional generation**, not trajectory rollout.

BONET vs. Guided Diffusion: Core Difference

Key conceptual distinction

BONET is process-centric: it asks *how do we get there?*
Guided diffusion is distribution-centric: it asks *what does a high-value solution look like?*

Example: "learning the path" vs. "learning the destination"

Suppose we observe offline evaluations

$$\boldsymbol{x}_1 \mapsto 20, \qquad \boldsymbol{x}_2 \mapsto 35, \qquad \boldsymbol{x}_3 \mapsto 55, \qquad \boldsymbol{x}_4 \mapsto 70, \qquad \boldsymbol{x}_5 \mapsto 85.$$

Our goal is to propose new candidates with even higher value.

How BONET thinks

It organizes data into something like an optimization trajectory:

$$20 \to 35 \to 55 \to 70 \to 85$$

Then it learns:

- given the previous points,
- and given how much regret budget remains,
- what a plausible **next improving step** should be.

BONET is learning the **path of improvement**.

How guided diffusion thinks

It does not ask for the next step in a trajectory. Instead, it learns:

- what kinds of $\boldsymbol{x}$ are associated with **high values**,
- and how to sample such $\boldsymbol{x}$ by denoising from noise.

Guided diffusion is learning the **shape of the high-value region**.

Conclusion

Thank you for your attention.

Questions and discussion

References

S. M. Mashkaria, S. Krishnamoorthy, and A. Grover.
Generative Pretraining for Black-Box Optimization.
Proceedings of the 40th International Conference on Machine Learning (ICML), 2023.

S. Krishnamoorthy, S. M. Mashkaria, and A. Grover.
Diffusion Models for Black-Box Optimization.
Proceedings of the 40th International Conference on Machine Learning (ICML), 2023.

D. H. Brookes and J. Listgarten.
Design by Adaptive Sampling.
arXiv preprint arXiv:1810.03714, 2018.

D. H. Brookes, H. Park, and J. Listgarten.
Conditioning by Adaptive Sampling for Robust Design.
Proceedings of the 36th International Conference on Machine Learning (ICML), 2019.