Test-time Scaling of Diffusion Models via Black-Box Optimization

Status	Backlog
Owner	Jonathan Wenger
Role	Advisory
Tags	

In a Nutshell

Background

-

🚨 **Problem**

-

🔭 **Observation(s)**

-

💡 **Idea**

-

Method

ϵ-greedy search

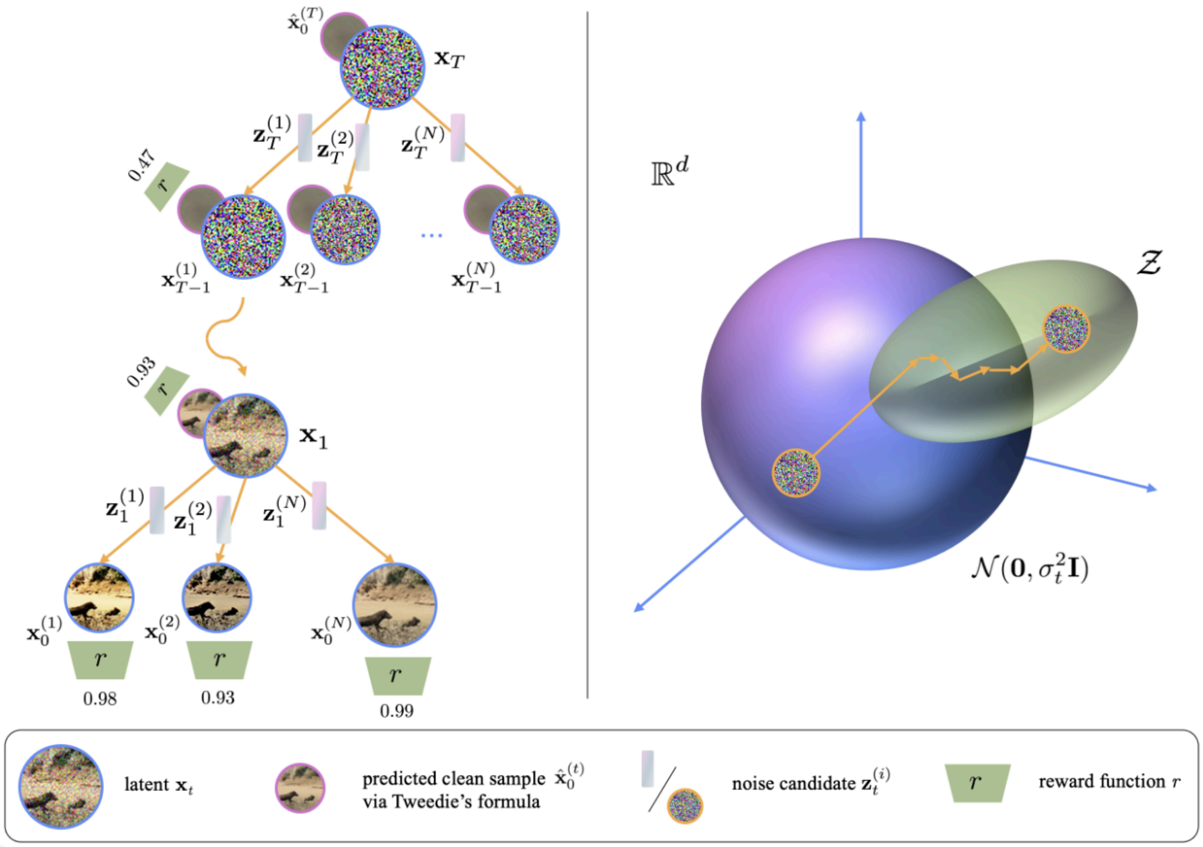


Figure 1: (*Left*) Implicit denoising tree traversed by search algorithms. (*Right*) Visualization of local search in noise space to maximize reward at a single timestep t.

Algorithm 1 ϵ-greedy noise search

Require: Discretization timesteps T, context (e.g. class label) $\mathbf{c}$, learned denoising network D_θ, max. step size scaling factor λ, number of local search iterations K, branching factor (number of noise candidates) N, sampling step function f, mixture proportion ϵ, reward function r, initial noise sample $\mathbf{x}_T$

1: **for** $t = T$ to 1 **do**

  Sample $\mathbf{p} \sim \mathcal{N}(\mathbf{0}, \sigma_t^2\mathbf{I})$ // pivot

2: **for** $k = 1$ to K **do**

  $\forall\ i = 1, \ldots, N$ sample noise candidate $\mathbf{z}_t^{(i)}$ as follows: with probability ϵ sample $\mathbf{z}_t^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma_t^2\mathbf{I})$, else let $\mathbf{z}_t^{(i)} = \mathbf{u}\sqrt{2d}\mathbf{z} + \mathbf{p}$ where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \mathbf{u} \sim \text{Unif}(0, \lambda)$

  Set $\mathbf{p} = \mathbf{z}_t^{(i)}$ for i s.t. one-step $\hat{\mathbf{x}}_0$ prediction using $\mathbf{z}_t^{(i)}$ attains highest score under r

3: **end for**

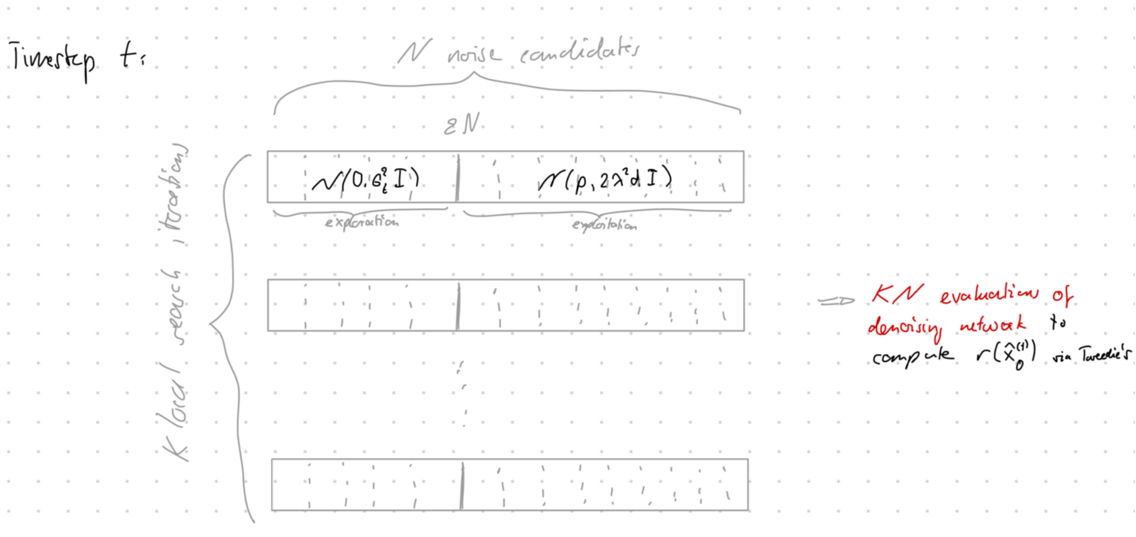
  Set $\mathbf{x}_{t-1} = f(\mathbf{x}_t, t, \mathbf{c}, \mathbf{p})$

4: **end for**

5: **return** $\mathbf{x}_0$

This simple change from the neighborhood in zero-order search to the ϵ-contaminated mixture distribution

$$\epsilon\mathcal{N}(\mathbf{0}, \sigma_t^2\mathbf{I}) + (1-\epsilon)\mathcal{N}(\mathbf{p}, 2\lambda^2 d\mathbf{I})$$



Problem: ϵ-greedy search is too expensive

- requires TNK number of function evaluations (NFEs) instead of T for naive sampling

Table 3: **NFE formulas by sampling method using T timesteps.** Note that NFEs denote "number of function evaluations" (i.e., number of calls to D_θ) for sampling a single image.

Method	NFE formula
Naive Sampling	T
Best of N Sampling	NT
(N, B)-Beam Search	$(N+B)T$
(N, S)-MCTS	$(N+S)T^2$
(N, K)-Zero-Order Search	NKT
(N, K)-ϵ-greedy	NKT

Table 1: **EDM results by sampling method.** Each value in columns 2-4 is obtained by generating 36 images given random ImageNet class labels. The column header denotes the reward used both to score the final images and during sampling if applicable. We set $\lambda = 0.15$ and $\epsilon = 0.4$. Note that we measure distance as $\|\mathbf{z}_t^{(i)} - \mathbf{p}\|_F$, hence let λ a scaling factor applied to $\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\mathcal{N}(0,1)^d}[\|\mathbf{x}-\mathbf{y}\|_F] \approx \sqrt{2d}$ (where $\|\cdot\|_F$ is the Frobenius norm. Re. notation, N is branching factor, B is beam width, S is number of MCTS simulations, and K is the number of local search iterations. Generating each sample took <1 second for naive sampling (the lowest-compute method) and <1 minute for MCTS (the highest-compute method) on a single A100 (40GB). We provide error bars for each reward and sampling method, computed as the standard deviation of scores for a given prompt over 20 generations with variability from randomness of the noise draws.

Method	Brightness	Compressibility	Classifier	NFEs
Naive Sampling	0.4965±0.01	0.3563±0.07	0.3778±0.04	18
Best of 4 Sampling	0.5767±0.01	0.4220±0.02	0.5461±0.00	72
Beam Search ($N=4, B=2$)	0.6334±0.02	0.4679±0.05	0.5536±0.02	144
MCTS ($N=4, S=8$)	0.7575±0.02	0.5395±0.04	0.9666±0.03	3888
Zero-Order Search ($N=4, K=20$)	0.6083±0.01	0.3751±0.02	0.6261±0.04	1440
ϵ-greedy ($N=4, K=20$)	**0.9813**±0.01	**0.7208**±0.03	**0.9885**±0.04	1440

Observation

- Exploitation is only useful at intermediate timesteps

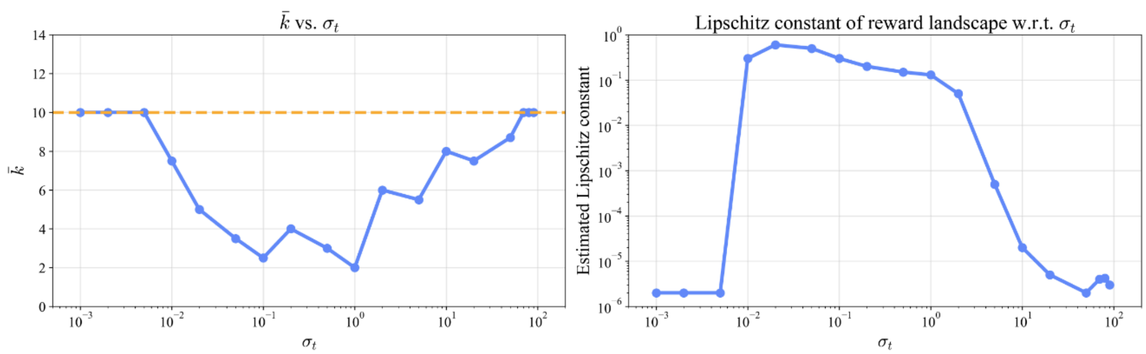


Figure 2: *(Left)* Average local-search iteration at which a random Normal candidate is chosen ($\bar{k}$) as a function of σ_t. *(Right)* Estimated Lipschitz constant of the ImageNet reward as a function of $\mathbf{x}_t$, across σ_t values.

Note: $\sigma(0) = 0$ and $\sigma(T) \gg 0$, where $t = 0$ corresponds to a clean image.

see large returns from hill-climbing. Indeed, we see this empirically: for each local search iteration $k \in \{1, \ldots, K\}$ of ϵ-greedy, let $I_k = \mathbb{I}$(ϵ-greedy selects one of the random Normal draws instead of all the neighborhood draws in iteration k). Define $\bar{k} = \sum_{k=1}^{K}(k-1)I_k$. Note that if ϵ-greedy chooses one of the random draws at every iteration, $\bar{k} = 10$; but if $\bar{k} < 10$, ϵ-greedy only chooses the random draw in early iterations k and only hill-climbs (chooses candidates from the neighborhood) at later iterations. Figure 2 (left) plots $\bar{k}$ as a function of σ_t. For initial denoising steps, the random noise is always chosen, hence $\bar{k} = 10$. But in the intermediate denoising steps, $\bar{k} \ll 10$—supporting our hypothesis that we initially need a couple of random draws to get to $\mathcal{Z}$, but for all subsequent iterations, *hill climbing is required for us to potentially leave the Normal distribution and find noises that maximize the reward.*

- The above graph shows that for $t \approx 0$ and $t \approx T$ we are wasting 40% of noise samples (and thus NFEs) on exploitation, while for $t \approx \frac{T}{2}$ we are wasting almost 60% of noise samples (and thus NFEs) on hill climbing.

Naive Idea

J.1 Compute Efficiency

For N noise candidates and K local search iterations per timestep, the ϵ-greedy approach requires NK times the NFEs compared to vanilla sampling for generating a single image. While this approach remains computationally linear with respect to the number of timesteps T, thus making it significantly less costly than more computationally intensive alternatives like MCTS, it nonetheless poses a computational burden. This additional computational cost could limit practical deployment scenarios, particularly where resources or latency constraints are critical factors.

To mitigate this, recalling Figure 2, as a proof-of-concept, we run zero-order search and ϵ-greedy with $K = 20$ for only $\{t : 0.01 \leq \sigma_t \leq 1\}$, $K = 1$ otherwise. This yields rewards within ± 0.04 of the original results (where $K = 20$ is used at all denoising steps), but cuts the NFEs by more than half. We recognize exploration of similar approaches to increase computational efficiency as an important direction for future research.

Improved Idea?

- Improve upon ϵ-greedy search by learning when to explore vs exploit as a function of time / noise schedule
  - collect statistics on observed rewards and when pivot was changed from previous generated images / proteins
- Instead of always with fixed probability $\epsilon = 0.4$ sampling from a Gaussian, learn when to explore and when to exploit.
  - ▼ *Idea*: learn to stop exploiting early if it is not useful.
    - based off of their proof-of-concept experiment, large K is only useful if we can exploit
    - learn distribution over $\bar{k}(t)$ and draw K_t for each timestep at random from that distribution after that. ⇒ fewer NFEs

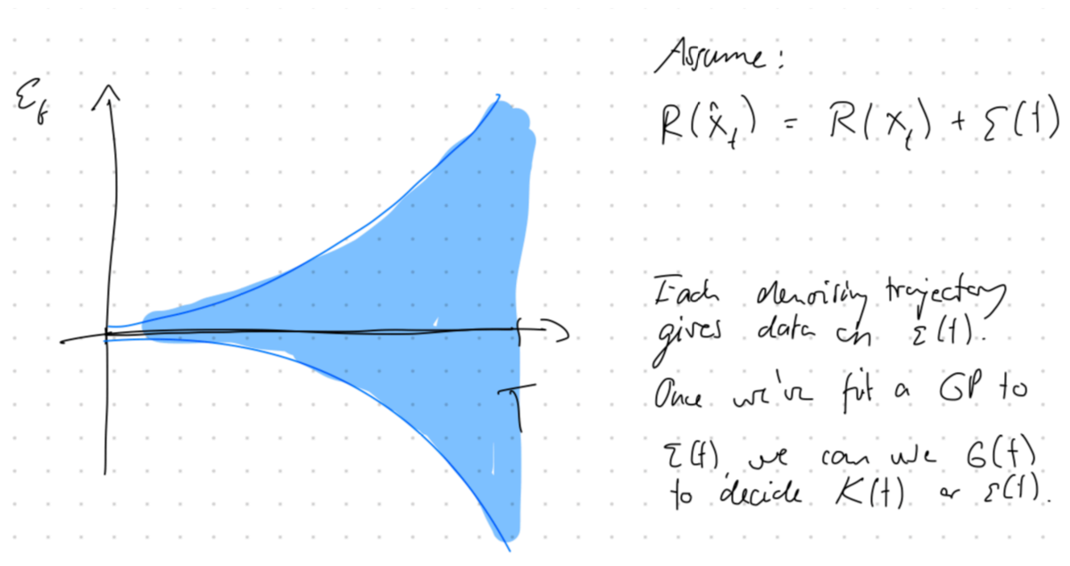
- Concretely: infer $p(t)$ of $\bar{k}(t) \sim Bin(K, p(t))$ from denoising multiple initial images and observing $\bar{k}(t)$. After that, simply draw $\bar{k}(t)$ or use $\text{ceil}(Kp(t))$

▼ *Idea*: learn to exploit more if useful

- define $\epsilon(t)$ using $\bar{k}(t)$ for an adaptive exploration vs exploitation trade-off ⇒ higher reward
    - $\bar{k}(t) = K \Rightarrow \epsilon(t) = 1$
    - $\bar{k}(t) \approx 0 \Rightarrow \epsilon(t) \approx 0$
- use data on $\bar{k}(t)$ from initial images.

▼ *Idea*: learn how good the reward model is as a function of time

- learn $R(t)$ and only use large number of samples Kor exploitation probability $(1-\epsilon)$ if uncertainty in $R(t)$ is small.
- Use GP with non-stationary kernel to learn $R(t)$ from (noisy) observations
    - $\epsilon(t) \approx R(\hat{x}_t) - R(x_0)$



Experiments

Next Steps

Literature

- https://arxiv.org/abs/2506.03164