NucleoBench: A Large-Scale Benchmark of Neural Nucleic Acid Design Algorithms

Joel Shor [1] **Erik Strand** [1 2] **Cory Y. McLean** [3]

Abstract

One outstanding open problem with high therapeutic value is how to design nucleic acid sequences with specific properties. Even just the 5' UTR sequence admits 2×10^{120} possibilities, making exhaustive exploration impossible. Although the field has focused on developing high-quality predictive models, techniques for generating sequences with desired properties are often not well benchmarked. Lack of benchmarking hinders production of the best molecules from high-quality models, and slows improvements to design algorithms. In this work, we performed the first large-scale comparison of modern sequence design algorithms across 16 biological tasks (such as transcription factor binding and gene expression) and 9 design algorithms. Our benchmark, NucleoBench, compares design algorithms on the same tasks and start sequences across more than 400K experiments, allowing us to derive unique modeling insights on the importance of using gradient information, the role of randomness, scaling properties, and reasonable starting hyperparameters on new problems. We use these insights to present a novel hybrid design algorithm, AdaBeam, that outperforms existing algorithms on 11 of 16 tasks and demonstrates superior scaling properties on long sequences and large predictors. Our benchmark and algorithms are freely available online.

1. Introduction

Artificial intelligence (AI) has the potential to dramatically improve the process of therapeutic drug discovery. Designing nucleic acid sequences (DNA and RNA molecules) with specific properties is a critical challenge for multiple stages of the drug development pipeline (Laganà et al., 2015; Shen et al., 2024; Lin et al., 2024; Sadybekov & Katritch, 2023). Solving it could transform how we approach therapeutics: enabling more precise CRISPR guide RNAs with minimal off-target effects for gene editing therapies (Tan et al., 2022); creating mRNA vaccines with optimized stability and translational efficiency (Zhang et al., 2023); designing antisense oligonucleotides that maximize target binding while minimizing immunogenicity (Lin et al., 2024; Hwang et al., 2024); and developing aptamers with enhanced specificity for diagnostic applications (Doench et al., 2016). Currently, each of these applications requires years of wet lab optimization and hundreds of millions of dollars in costs (Zhang et al., 2023; DiMasi et al., 2016; Paul et al., 2010). Computational sequence optimization could collapse these timelines from years to days and significantly reduce costs by identifying high-performing candidate sequences prior to experimental validation (Sadybekov & Katritch, 2023; Mouchlis et al., 2021). Although the field has developed high-quality predictive models of nucleic acid properties (Churkin & Barash, 2025; Gosai et al., 2024; Avsec et al., 2021a), the algorithms that generate optimized sequences from these models have received significantly less attention. One reason is that there are fewer benchmarks to understand sequence design algorithms. This gap hinders our ability to produce therapeutic-grade molecules even when using state-of-the-art predictive models.

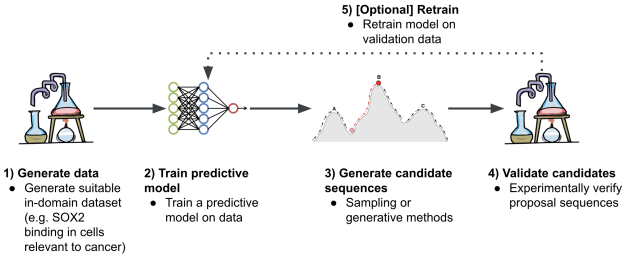

Figure 1. Graphical representation of *in silico* nucleic acid design.

Current efforts to design novel nucleic acid sequences often involve four core steps (Figure 1): 1) Generate data to collect a high quality dataset that has desired properties (properties such as cell-type specific expression, binding affinity to a particular transcription factor, translational effi-

[1]Move37 Labs [2]MIT Center for Bits and Atoms [3]Google Research. Correspondence to: Joel Shor <joel.shor@move37labs.ai>.

Proceedings of the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

Table 1. Nucleic acid design from sequence benchmarks. All benchmarks prior to NucleoBench are limited either in the range of tasks they measure against, the range of optimizations they compare, or the complexity of the task.

NAME	YEAR	ALGOS	TASKS	SEQ. LENGTH (BP)	DESIGN BENCHMARK	LONG SEQS	LARGE MODELS	PAIRED START SEQS.
Fitness Landscape Exploration Sandbox	2020	4-6	9	Most 10, all <100	✔	✗	✗	✔
Computational Optimization of DNA Activity	2024	3	3	200	✔	✗	✗	✔
gRelu	2024	2	5	500K (20 edit)	✗	✗	✔	✗
Linder et al repos	2021	2	20	<600	✔	✗	✗	✗
NucleoBench (ours)	2025	9	16	256-3K	✔	✔	✔	✔

ciency, half-life, etc), 2) Train a predictive model, usually a neural network, to predict the property from sequence data, 3) Generate candidate sequences using the trained model and a sequence design algorithm, 4) Validate candidates by synthesizing the sequences and measuring their properties, and optionally use the evaluated candidates to retrain a new model for further evaluation.

The academic literature has focused primarily on steps 1, 2, and 4, often using a core set of established algorithms for step 3. While this has produced important progress in benchmarks (Table 1) and optimization algorithms (Table 2), they both remain areas with substantial opportunities for innovation.

First, progress in the field could be accelerated by addressing the current fragmentation of benchmarking efforts. Table 1 shows a list of benchmarks since 2020. Sinai et al. (2020) has an impressive range of tasks, but does not include modern tasks (since 2021), and is limited in complexity of the number of sequences per task, with most containing around ten sequences and all fewer than 100. Lal et al. (2024) is an actively maintained framework, though as of this writing it only includes three optimization algorithms, limiting its utility for comprehensive algorithm assessment. Benchmarks on adjacent problems, such as Mathews (2019) and Badura et al. (2025), do not necessarily transfer to sequence-only design.

Second, expanding the set of optimization strategies evaluated could accelerate progress. Our analysis of the benchmarks since 2020 (Table 1) shows that all but one still use simulated annealing (Van Laarhoven & Aarts, 2010) or genetic algorithms as a primary optimization strategy, despite these methods predating modern deep learning architectures by decades. Simulated annealing, developed in the 1970s, remains featured in leading RNA design textbooks (Churkin & Barash, 2025), while variations of genetic algorithms dominate most benchmarks in Table 1 (Gosai et al., 2024; Sinai et al., 2020). These algorithms, while simple and general, cannot take advantage of gradient information from neural networks. Notable exceptions like Ledidi (Schreiber et al., 2020) and FastSeqProp (Linder & Seelig, 2021) incorporate gradient information but still employ greedy search strategies that can converge to suboptimal solutions for complex fitness landscapes.

In this work, we propose NucleoBench, the largest comparison of nucleic acid design algorithms to date. We conducted over 400K design experiments to compare five of the most commonly used sequence design algorithms (hereafter "designers") across 16 different tasks. As a result of our analysis, we propose four other designers that combine the strengths of multiple categories. One of them, AdaBeam, is the best performing designer, and outperforms the others on 11 of the 16 tasks. Our benchmark code is public. To summarize, our main contributions are:

1. We introduce NucleoBench, the largest nucleic acid design benchmark to date, including tasks with "large" models and longer design sequences.
2. We comprehensively evaluate both standard and novel designers across 16 tasks by running over 400K experiments.
3. We provide data-driven answers to questions such as "what are reasonable starting parameters for new tasks," "how sensitive are the designers to hyperparameters," "how do designers scale with model size and sequence length," and "what is the role of start sequence and random seed on designer performance".
4. Using insights from #1-3, we introduce AdaBeam, a new designer that outperforms the others, and has improved scaling properties.

2. Methods

Each experiment contains four inputs: 1) design task, 2) designer, 3) designer hyperparameters, and 4) start sequence. We review these below.

Table 2. Summary of designers in NucleoBench. Above the solid line are designers already found in the nucleic acid design literature. Below the line are designers from the search literature not previously used to benchmark nucleic acid sequence design and hybrid algorithms devised in this work.

Algo	Description	Gradient-based
Directed Evolution	Random mutations, track the best.	✗
Simulated Annealing	Greedy optimization with random jumps.	✗
AdaLead	Iterative combining and mutating of a population of sequences.	✗
FastSeqProp	Sampling and the straight-through estimator for maximal input.	✔
Ledidi	Sampling and the gumbel softmax estimator for maximal input.	✔
Ordered Beam	Greedy search, in fixed sequence order, with cache.	✗
Unordered Beam	Greedy search with cache.	✗
Gradient Evo	Directed Evolution, guided by model gradients.	✔
AdaBeam	Hybrid of Unordered Beam and improved AdaLead.	✗

2.1. Design tasks

We benchmark designers on 16 tasks in three categories. The summary of tasks can be found in Table 3, and details can be found in the Appendix "Design tasks".

Task selection: Our tasks capture the diversity of nucleic acid design challenges. These tasks collectively span multiple dimensions of complexity: sequence length (from 200bp to 196,608bp), model inference speed (from 5ms to 15s per example), and biological context (from localized transcription factor binding to cell-type-specific gene expression). We prioritized tasks with established predictive models and biological relevance, particularly focusing on models that represent key regulatory mechanisms in the genome.

The task categories evaluate different architectures (CNNs for BPNet and Malinois, attention mechanisms for Enformer). To our knowledge, NucleoBench marks the first time that Enformer was used directly for a design task. While Enformer is widely used for discriminative tasks, its size has made it computationally difficult to use for design tasks. For more details, see the Appendix "Design Tasks."

2.2. Sequence designers

Our study compares five designers from the literature, two from the computer science search literature that to our knowledge have not been previously applied to nucleic acid design, and two novel hybrid designers. Designers attempt to optimize a sequence's score on an objective function given by the design task model. Designers based on black-box and "gradient-based" (uses model gradients) methods are evaluated. Benchmarking on identical start sequences improves statistical power and allows us to quantify performance variance due to randomness and start sequence.

Baseline algorithm selection: The five baseline designers were chosen to be representative of different kinds of algorithms, and to reflect what has been used in prior benchmarks. Directed Evolution and AdaLead are common examples of greedy evolutionary algorithms. They are easy to implement, and have been used in previous benchmarks (Gosai et al., 2024; Sinai et al., 2020; Lal et al., 2024). FastSeqProp and Ledidi are examples of strong performing, greedy, purely gradient-based approaches. They are also commonly used in benchmarks (Gosai et al., 2024; Sinai et al., 2020; Lal et al., 2024; Linder & Seelig, 2021; 2019). Simulated annealing is the only non-greedy optimizer, and is a common baseline for works in this field (Gosai et al., 2024; Sinai et al., 2020; Lal et al., 2024; Linder & Seelig, 2021; 2019). As discussed in the section "Comparison to generative modeling", we excluded generative models in this version of NucleoBench.

2.2.1. Ordered and Unordered Beam search

We introduce **Ordered and Unordered Beam search** to nucleic acid design. These designers are staple search algorithms in the computer science literature, and are often used in speech processing. The "beam" in these designers allows them to track multiple strong candidate sequences rather than just the single best sequence. We compare these designers to understand their performance and *the impact of fixing edit order*.

For both beam search designers, we decompose edit selection into two independent steps: selecting which location to edit, then selecting what the edit should be.

$$\begin{aligned}\Pr[\text{bp at location } n_t \text{ is changed to } m|H_{t-1}] = \\ \Pr[m|\text{edit location } n_t]\cdot \\ \Pr[\text{edit location } n_t|H_{t-1}]\end{aligned}$$

where H_{t-1} is the relevant history of edits that the algorithm has made for the first $t-1$ edits. For Ordered Beam, we select a random permutation of edit order at the start of the algorithm, and proceed to make edits in that order. The

Table 3. Summary of design tasks in NucleoBench. *Input length is 200K, but only 256 bp are edited. **All models are 55ms except the ATAC model, which is 260.

TASK CATEGORY	MODEL	DESCRIPTION	NUM TASKS	SEQ LEN (BP)	SPEED (MS / EXAMPLE)
Cell-type specific cis-regulatory activity	Malinois	How DNA sequences control gene expression from the same DNA molecule. Cell types are: precursor blood cells, liver cells, neuronal cells.	3	200	2
Transcription factor binding	BPNet-lite	How likely a specific transcription factor (TF) will bind to a particular stretch of DNA.	12	3000	55 / 260**
Selective gene expression	Enformer	Prediction of gene expression.	1	196,608 / 256 *	15,000

designer stops when all locations have been edited once. Note that this allows any start sequence to be edited to any other sequence. Thus, if l_i is our predetermined edit order, then for Ordered Beam, we have:

$$\Pr[\text{edit location } n_t | H_{t-1}] = \begin{cases} 1 & \text{if } n_t = l_t \\ 0 & \text{otherwise} \end{cases}$$

For Unordered Beam search, we select a single location uniformly at random to edit. Thus, for Unordered Beam:

$$\begin{aligned} \Pr[\text{edit location } n_t | H_{t-1}] &= P(\text{edit location } n_t) \\ &= 1/\text{sequence length} \end{aligned}$$

For both algorithms, the new nucleotide is selected uniformly at random to ensure that it is modified.

2.2.2. ANALYSIS OF BASELINES AND MOTIVATION FOR NOVEL DESIGNERS

Efficient gradient calculation on large models: Ledidi implementations from (Gosai et al., 2024; Lal et al., 2024; Schreiber et al., 2020) and FastSeqProp (Linder & Seelig, 2021) do not scale to sequences of 100 kilo basepairs on Enformer, due to backpropagation on the large model causing out-of-memory errors. This is because these designers (in PyTorch) use the common technique of "gradient masking." While this gives the correct result, it does not reduce peak memory usage. In contrast, NucleoBench optimizers are able to perform backpropagation with the Enformer model by using a more memory-efficient technique that we call "gradient concatenation." Roughly speaking, while gradient masking multiplies the input by a constant-Tensor mask of 0s and 1s, gradient concatenation constructs the input Tensor from two sub-Tensors, only one of which is marked for gradients. Gradient concatenation is more memory efficient when the backwards pass is much more memory intensive than the forward pass. For more details, see the NucleoBench implementation or the Appendix section "Gradient Concatenation."

Identifying limitations in AdaLead to motivate novel sequence designers: AdaLead was the best performing non-gradient baseline designer (Figure 2), so we ran ablation studies to understand which components of AdaLead were most important. We found a number of unexpected results. First, AdaLead without recombination always outperformed AdaLead with recombination (this was also observed in Gosai et al. (2024)).

Second, we theoretically proved and empirically verified that the reference implementation of AdaLead performs an $O(n)$ computation with rejection sampling where there is an identical $O(1)$ computation. Rejection sampling induces the following distribution on the number of edits per step:

$$\Pr[N = n] = \frac{\text{Binomial}(n, l, m')}{1 - (1 - m')^l} \quad \text{if } 1 \leq n \leq l,$$
$$0 \text{ otherwise}$$

$$m' := \frac{3}{4}m, \; m := \text{The "mutation rate" of the algorithm}$$
$$l := \text{length of the sequence}$$

Given this closed-form solution, we can replace the expensive rejection sampling process with a single sample from this distribution.

Third, unlike other algorithms, the rejection sampling can cause an infinite loop during optimization.

2.2.3. GRADIENT EVO: HYBRID OF DIRECTED EVOLUTION AND GRADIENT METHODS

We introduce a novel designer, gradient-guided Directed Evolution (Gradient Evo), which is a hybrid between the simplicity of Directed Evolution and the performance of Ledidi and FastSeqProp. The algorithm uses Directed Evo-

lution, where edit locations are guided by Taylor *in silico* mutagenesis (TISM) (Sasse et al., 2024). TISM approximates the effect of changing nucleotides in a sequence in a computationally efficient way. The equation for TISM, taken from Sasse et al. (2024) is:

$$ISM(s_0', l, b_l) \approx \frac{df^{(l,b_1)}}{ds_0} - \frac{df^{(l,b_0)}}{ds_0} = TISM(s_0', l, b_1)$$

$s_0' :=$ nucleotide sequence

$b_0 :=$ the nucleotide at location l in the original sequence

$b_1 :=$ the nucleotide at location l in the modified sequence

2.2.4. AdaBeam: Adaptive Beam Search

Inspired by our analyses of AdaLead and Beam Search, we introduce **Adaptive Beam search (AdaBeam)**. As in beam search, we decompose each round into an edit location selection step and an edit step. Unlike AdaLead's edit location step, which implicitly repeatedly performs an O(n) computation, AdaBeam's location selection step performs a single O(1) computation. On comparable hyperparameters, AdaBeam is able to compute steps on 3,000 nucleotide sequences roughly two times faster than AdaLead due to explicit sampling. We describe the full algorithm in the Appendix.

2.3. Designer hyperparameters

The hyperparameters are designer-specific. These are factors such as "learning rate" in FastSeqProp and Ledidi, and "mutation rate" in Directed Evolution.

Hyperparameter selection: We used hyperparameter values that perform well in other benchmarks (Gosai et al., 2024; Sinai et al., 2020; Lal et al., 2024; Linder & Seelig, 2019), as well as adaptations (e.g. lower mutation on problems with longer sequences).

2.4. Start sequences

The role of start sequences in designers is poorly understood, in part due to the difficulty of running large-scale experiments with identical starting sequences. We run each experiment in a task on **100 identical starting sequences**, allowing us to evaluate algorithms fairly across the same sequence. This allows us to have greater statistical power by performing paired statistical tests instead of unpaired ones, such as the Friedman test (Friedman, 1937), as well as explore the idea of "difficult start sequences," which are start sequences that are difficult across all designers. See the Appendix for how start sequences were chosen.

Start sequence selection: For all tasks, start sequences are picked using three different methods, according to which is used in the literature: 1) random sequence, 2) random sequence flanked by a real sequence, 3) real sequence mined from the human genome having specific properties. Details are in the Appendix "Design tasks".

3. Evaluation

3.1. Experimental setup

Each experiment contains four inputs: 1) design task, 2) designer, 3) designer hyperparameters, and 4) start sequence, and experiments are independently parallelizable. We ran each experiment for a fixed period of time (12 hours for Enformer jobs, 8 hours for others) or until a designer-specific end condition was met (e.g., Ordered Beam finishes when every sequence location has been mutated). We ran all experiments on CPU-only machines, primarily type `n1-highmem-16`, using Google Batch.

3.2. Metrics

The primary evaluation metric we used was the final fitness of the designed sequence after the experiment, according to the task model. For example, this might be the binding affinity of GATA2 to the sequence, according to the GATA2 BPNet model. Other metrics that we collected are the time series of energy values throughout optimization (collected every N steps, where N varies depending on the designer), and the time taken for each optimization step.

3.3. Methodology of analyses

Performance: We ran over 400K experiments to determine the best performance for each algorithm on each task. The results are shown in Figure 2. We selected hyperparameters using the methodology described in section "Designer hyperparameters". The same starting start sequences were used for each designer in a given task (see section on 'Start Sequence' for more details). The 95% confidence interval was computed using Student's t-distribution of the final energy across start sequences and random seeds.

Performance variability from random seed: To assess variability from algorithmic randomness, each experiment was rerun with five different random seeds using the best out-of-the-box hyperparameters. For each combination of design task, algorithm, hyperparameter, and start sequence, we computed the variance across these five seeds, and then determined the median of these variances over the 100 start sequences. The median variance for each (design task, designer) pair is detailed in the Appendix section "Performance variability due to random seed." For a comparable, single score per designer, we aggregated these results using a non-parametric, 0-based order score derived from its variance rank on each task, averaged across all tasks.

Performance variability from start sequence: To investi-

gate the underexplored impact of starting sequences, a previously computationally challenging analysis, NucleoBench ran each design task, algorithm, and hyperparameter combination on 100 identical start sequences. We quantified the effect of start sequence on each optimizer by computing the variance of final energies across these 100 starts using the best hyperparameters for each (task, algorithm) pair, with detailed results in the Appendix section "Performance variability based on start sequence." To aggregate this across tasks into a single score per algorithm, we employed non-parametric, 0-based order scores based on variance ranks, averaged across tasks.

Existence of "difficult start sequences": The previous analysis quantifies the effect of start sequence on the quality of designed sequences found by the designer. One outstanding question is how correlated these performance fluctuations are with the start sequence, and whether there exist "intrinsically difficult start sequences." To that end, we analyzed whether certain sequences are likely to be responsible for poor performance in an optimization-agnostic way. For each task, we look at the experiment results from the best hyperparameters for each algorithm. We perform the non-parametric Friedman Test and the Nemenyi posthoc test by treating the 100 start sequence identities as nominal variables, the different designers as treatments, and the performances as continuous Y variables.

Convergence times: To assess optimizer convergence speed, we measured the time required for each (design task, designer) combination, using the best out-of-the-box hyperparameters, to reach performance within 0.05 of its final best score for that run. The median of these times across different random seeds was computed. For a single, comparable score, we aggregated these convergence times per designer using a non-parametric, 0-based order score based on its rank for each task (where faster convergence ranked better), averaged across all tasks.

4. Results

4.1. Per-task performance analysis

Figure 2 shows the distribution of final scores achieved by each optimization algorithm. The best hyperparameters were optimized for each task individually (see Appendix section "Per-task results"). Scores are aggregated across tasks. AdaBeam was on average the most performant algorithm (paired t-test across all tasks, paired on (task, start sequence), $p = 7.6 \times 10^{-23}$, $t = 9.9$, $df = 2798$), with it and Ledidi the clearly most performant designers. Each task had at least one of those two designers as the best performing; AdaBeam was the best or second best on 15 of the 16 tasks evaluated and Ledidi was the best or second best on 14 of 16. Indeed, on the BPNet and Enformer modeling tasks,

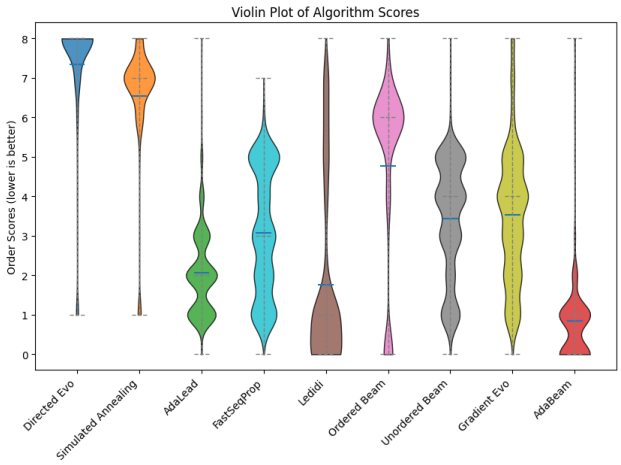

Figure 2. The distribution of final scores for each algorithm.

which showed substantial performance differentiation between designers, only the OTX1 task had a distinct designer as the second best (Table A.1). Tasks demonstrated distinct difficulty levels, with all Malinois tasks being solved by all designers except Directed Evolution, Simulated Annealing, and Gradient Evo, and a larger spread of performance on BPNet and Enformer tasks (Table A.1).

4.2. Performance Variability and Convergence Speed Analysis

Performance variability based on random seed: Table 4 (top) shows the aggregate performance variability based on random seed according to the non-parametric order score. All designers have roughly comparable variance attributable to random seed except for simulated annealing, which is markedly lower. The relatively small sensitivity to random seed is consistent across tasks, with Enformer tasks contributing the largest magnitude differences in the 25th-75th percentile performance (Table A.2).

Performance variability based on start sequence: The role of starting sequence in nucleic acid designers has been underexplored, mostly due to the computational difficulty of running such an analysis. In NucleoBench, we were able to run this analysis fairly because each designer was started with the same collection of 100 sequences. The non-parametric order scores in Table 4 (middle) indicate the variances based on starting sequence. The lower variability of simulated annealing may stem at least in part due to its consistently poor performance across tasks (Table A.3).

Existence of "difficult start sequences": Table 5 shows that intrinsically difficult start sequences exist, and are not uniformly distributed between tasks. Specifically, we observed roughly four categories of problem: no difficult seeds (8/16), 1 difficult seed (6/16), more than 1 and fewer than

Table 4. Top) Variability due to start sequence. Lower is better (lower variance). Middle) Variability due to random seed. Lower is better (lower variance). Bottom) Relative order of designer convergence time. Lower is better (faster convergence). See "Performance variability from start sequence" in Methods for a description of the order score.

Analysis	**Metric**	**Directed Evolution**	**Simulated Annealing**	**AdaLead**	**FastSeq Prop**	**Ledidi**	**Ordered Beam**	**Unordered Beam**	**Gradient Evo**	**AdaBeam**
Variability from random seed	Order score ↓	2.4	0.6	3.4	3.6	4.6	6.3	4.9	5.2	3.3
Variability from start sequence	Order score ↓	2.9	0.8	4.2	3.4	6.0	2.6	4.4	5.6	4.8
Convergence time	Order score ↓	7.5	5.7	3.6	3.3	1.5	3.0	3.1	6.8	1.3

5 difficult seeds (1/16), and many ($\geq 5\%$) difficult seeds (1/16). Cell-specific expression tasks and the majority of transcription factor binding tasks did not have any hard sequences.

Convergence: Table 4 (bottom) indicates aggregate convergence time per optimization algorithm. We find that Directed Evolution and its gradient-guided variant, Gradient Evo, are the slowest to converge, while AdaBeam and Ledidi are the fastest. Table 5 (bottom) shows the aggregate convergence time per task. Most tasks are similar in terms of convergence time, with the Enformer task taking the longest owing to its sequence length and model size, and OTX1 being a notable outlier that takes longer to converge on the BPNet model.

5. Discussion

Our analysis of the overall performance of multiple designers revealed several key takeaways. First, AdaBeam and Ledidi were the most performant designers both in terms of sequence optimization quality and convergence time. Their consistently strong performance indicate the robustness and effectiveness of their design strategies and suggests that these algorithms might generally be better at traversing the energy landscape of neural networks applied to problems in biology. AdaLead, FastSeqProp, Gradient Evo, and Unordered Beam were the next most performant designers, establishing a clear middle tier of performance. Interestingly, the Ordered and Unordered Beam designers were implemented similarly except for the fixed edit order, but the performance of Ordered Beam is noticeably inferior, suggesting that fixed-edit order is not a good strategy. Finally, Simulated Annealing and Directed Evolution, despite often being used as baseline designers, were the least performant designers benchmarked in this work. Neither designer could reliably solve the Malinois tasks, which were solved by all other designers except Gradient Evo. Researchers should consider not using these algorithms as baseline designers.

Task diversity is important to enable differentiation of designer performance. The three Malinois tasks were solved by nearly all designers; this performance saturation made differentiation difficult. On the other hand, the OTX1 task of BPNet yielded relatively poor performance for all designers and thus represents an opportunity for future improvement. The other 13 tasks showed meaningful performance differentiation across designers.

To our knowledge, NucleoBench is the first study to characterize designer performance variability due to random seed and start sequence. Encouragingly, we observed little performance variability attributable to random seed, lending support to the robustness of the different optimizer algorithms. Start sequence, on the other hand, is more important. While there were relatively small differences in the sensitivity to start sequences between designers on a particular task, the absolute difficulty of optimizing a sequence for a task does depend on its composition, with some intrinsically difficult start sequences discovered for Enformer and BPNet tasks.

Recent advances in model quality have stemmed in part from supporting longer input sequences. Designers need concomitant adaptation to support larger models and longer sequences. In this work, we limited to only 256 candidate positions to edit within sequences passed to the Enformer model, because Ledidi and FastSeqProp, the two most performant baseline designers, are gradient-based and could not optimize the full sequences passed to the Enformer-based tasks. Our results emphasize the importance of gradient-based algorithms, as gradient-based designers were the top or second-most performant designers across 15/16 tasks, and Gradient Evo significantly outperformed Directed Evolution. Improved designer support can arise from careful engineering and software design; here we showed that by switching from gradient masking, in which extraneous gradients are calculated and thrown away, to tensor concatenation, in which only relevant gradients are computed during backpropagation, we enabled Gradient Evo to scale to Enformer-based tasks far larger than those supported by Ledidi and FastSeqProp. This highlights the importance of reference implementations in the nucleic acid design space.

Finally, thorough ablation studies and careful experimental

Table 5. Top) Friedman test for the existence of extremal start sequences, then the Nemenyi post hoc test for identifying specific start sequences. Task name in the top row. Bottom row is (fraction of seeds that are extremal, Friedman test statistic, significance). Bottom) Aggregate amount of time for optimization convergence. Convergence times were computed for each task / optimizer, then converted to a z-score across tasks per optimizer. Z-scores were then averaged for each task. Higher scores indicate that this task took optimizers longer than others to converge. See "Performance variability from start sequence" in Methods for a description of the order score.

Task type		**Malinois**			**BPNet**												**Enformer**
Start sequence method		**Random**			**Mined from human genome**												**Mined from human genome**
		K562	**HepG2**	**SK-N-SH**	**ATAC**	**CTCF**	**E2F3**	**ELF4**	**GATA2**	**JUNB**	**MAX**	**MECOM**	**MYC**	**OTX1**	**RAD21**	**SOX6**	**DNase (↑ muscle ↓ liver)**
Existence of difficult start sequences	Fraction of external seeds (out of 100)	-	-	-	1	-	-	1	1	-	3	1	-	1	1	-	54
	Friedman test statistic	-	-	-	154	-	-	150	167	-	204	141	-	-	206	-	447
	Friedman test significance	-	-	-	3.4 E-4	-	-	7.6 E-4	2.4 E-5	-	3.0 E-9	3.4 E-3	-	-	1.8 E-9	-	1.2 E-45
Optimizer convergence	Order ↓	9.4	7.4	5.3	8.7	5.7	7.7	5.0	5.0	8.6	5.7	4.1	6.7	12.6	5.9	7.3	17.5

design enabled novel insights. We investigated AdaLead, the strongest non-gradient baseline designer, and observed that the best per-task results always had recombination and thresholding disabled (Appendix "Per-task results"). We also note that by performing paired statistical tests, we were able to use the Friedman test for ablation studies, and to identify difficult start sequences. Both analyses would have been much less sensitive without paired start sequences.

Our work has several limitations. First, the sequence designers we compared in NucleoBench are relevant to biology only insofar as the task models are accurate. While this work focused on the designers, there are important relationships between the quality of the task model and the performance of the designer that we did not explore. For example, we did not benchmark designers that are aware of model uncertainty. Second, we have not thoroughly compared how well the designers incorporate biological constraints or the biological plausibility of the optimized sequences. Third, we did not evaluate any generative models in this version of the benchmark.

Future work will expand NucleoBench to include more diverse tasks, including synthetic benchmarks, long-term planning challenges, and multi-objective optimizations, alongside deeper analysis of algorithm performance scaling with sequence length and model size. For optimization methods, priorities include developing algorithms aware of oracle uncertainty, explicitly managing exploration-exploitation tradeoffs, and adaptively balancing gradient and non-gradient information. Building on this work as a performance baseline, future efforts will also integrate biological plausibility constraints, either as fitness terms or filtering steps, which will be crucial for therapeutic relevance.

6. Conclusion

Work that improves the nucleic acid computational design phase that precedes costly wet lab validation has myriad therapeutic applications. For CRISPR therapeutics, gradient-informed algorithms like Ledidi could significantly improve guide RNA specificity by minimizing off-target effects, a major safety concern in gene editing therapies. In mRNA vaccine development, these optimized algorithms could enhance translational efficiency and stability, addressing key challenges that affected early COVID-19 vaccine storage requirements. For antisense oligonucleotide therapies targeting genetic disorders, gradient-guided sequence optimization provides a pathway to design sequences with higher target binding affinity while minimizing immunogenicity risks.

NucleoBench is an open-source benchmark of nucleic acid design optimization algorithms. As the most comprehensive standardized method for evaluating new designers, covering 16 different design tasks from a range of problem domains, NucleoBench represents a framework for accelerating progress in nucleic acid computational design. Through over 400K experiments, we empirically evaluated which designers consistently produce the highest-quality sequences, sets of reasonable hyperparameters to use for each designer, and elucidated the sensitivity of the tested designers to hyperparameters, start sequences, and random seeds, while confirming the critical role of gradients. The insights from this work suggest new avenues for improving designer performance and directly led to the development of AdaBeam, which outperformed existing designers on 11 of 16 tasks and possesses more favorable scaling properties. NucleoBench provides data for 16 common tasks as well as publicly available reference implementations of all designers. Further improvement is still needed, however, to handle even longer sequences and larger models, as well as to overcome the lim-

itations of greedy approaches. We hope that NucleoBench and AdaBeam spur further interest in the development of new optimization algorithms for nucleic acid sequence design.

Acknowledgments

We thank to Sager Gosai for his invaluable guidance on interpreting motifs and task selection. We thank Daniel Friedman for early discussions on formalizing edit order. We thank Anna Lewis and Vikram Agarwal for guidance throughout the paper writing process.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Adamczyk, B., Antczak, M., and Szachniuk, M. RNAsolo: a repository of cleaned PDB-derived RNA 3D structures. *Bioinformatics*, 38(14):3668–3670, July 2022.

Alamdari, S., Thakkar, N., van den Berg, R., Tenenholtz, N., Strome, R., Moses, A. M., Lu, A. X., Fusi, N., Amini, A. P., and Yang, K. K. Protein generation with evolutionary diffusion: sequence is all you need. *bioRxiv*, September 2023.

Anderson-Lee, J., Fisker, E., Kosaraju, V., Wu, M., Kong, J., Lee, J., Lee, M., Zada, M., Treuille, A., Das, R., and Eterna Players. Principles for predicting RNA secondary structure design difficulty. *J. Mol. Biol.*, 428(5 Pt A): 748–757, February 2016.

Avsec, Z., Agarwal, V., Visentin, D., Ledsam, J. R., Grabska-Barwinska, A., Taylor, K. R., Assael, Y., Jumper, J., Kohli, P., and Kelley, D. R. Effective gene expression prediction from sequence by integrating long-range interactions. *Nat. Methods*, 18(10):1196–1203, October 2021a.

Avsec, Z., Weilert, M., Shrikumar, A., Krueger, S., Alexandari, A., Dalal, K., Fropf, R., McAnany, C., Gagneur, J., Kundaje, A., and Zeitlinger, J. Base-resolution models of transcription-factor binding reveal soft motif syntax. *Nat. Genet.*, 53(3):354–366, March 2021b.

Badura, J., Zok, T., and Rybarczyk, A. Datasets for benchmarking RNA design algorithms. *Methods Mol. Biol.*, 2847:229–240, 2025.

Becquey, L., Angel, E., and Tahi, F. RNANet: an automatically built dual-source dataset integrating homologous sequences and RNA structures. *Bioinformatics*, 37(9): 1218–1224, June 2021.

Churkin, A. and Barash, D. (eds.). *RNA design: Methods and protocols*. Methods in molecular biology (Clifton, N.J.). Springer US, New York, NY, 2025.

Danaee, P., Rouches, M., Wiley, M., Deng, D., Huang, L., and Hendrix, D. bpRNA: large-scale automated annotation and analysis of RNA secondary structure. *Nucleic Acids Res.*, 46(11):5381–5394, June 2018.

DiMasi, J. A., Grabowski, H. G., and Hansen, R. W. Innovation in the pharmaceutical industry: New estimates of R&D costs. *J. Health Econ.*, 47:20–33, May 2016.

Doench, J. G., Fusi, N., Sullender, M., Hegde, M., Vaimberg, E. W., Donovan, K. F., Smith, I., Tothova, Z., Wilen, C., Orchard, R., Virgin, H. W., Listgarten, J., and Root, D. E. Optimized sgRNA design to maximize activity and minimize off-target effects of CRISPR-Cas9. *Nat. Biotechnol.*, 34(2):184–191, February 2016.

Eastman, P., Shi, J., Ramsundar, B., and Pande, V. S. Solving the RNA design problem with reinforcement learning. *PLoS Comput. Biol.*, 14(6):e1006176, June 2018.

Friedman, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Am. Stat. Assoc.*, 32(200):675–701, December 1937.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *arXiv [stat.ML]*, June 2014.

Gosai, S. J., Castro, R. I., Fuentes, N., Butts, J. C., Mouri, K., Alasoadura, M., Kales, S., Nguyen, T. T. L., Noche, R. R., Rao, A. S., Joy, M. T., Sabeti, P. C., Reilly, S. K., and Tewhey, R. Machine-guided design of cell-type-targeting cis-regulatory elements. *Nature*, 634(8036):1211–1220, October 2024.

Hitz, B., Kagda, M., Lam, B., Litton, C., Small, C., Sloan, C., Spragins, E., Tanaka, F., Whaling, I., Gabdank, I., Youngworth, I., Strattan, J. S., Hilton, J., Jou, J., Au, J., Lee, J.-W., Andreeva, K., Graham, K., Lin, K., Simison, M., Jolanki, O., Sud, P., Assis, P., Adenekan, P., Miyasato, S., Zhong, W., Luo, Y., Myers, Z., and Cherry, J. Data navigation on the ENCODE portal. *arXiv [q-bio.GN]*, July 2023.

Hwang, G., Kwon, M., Seo, D., Kim, D. H., Lee, D., Lee, K., Kim, E., Kang, M., and Ryu, J.-H. ASOptimizer: Optimizing antisense oligonucleotides through deep learning for IDO1 gene regulation. *Mol. Ther. Nucleic Acids*, 35 (2):102186, June 2024.

Ji, Y., Zhou, Z., Liu, H., and Davuluri, R. V. DNABERT: pre-trained bidirectional encoder representations from transformers model for DNA-language in genome. *Bioinformatics*, 37(15):2112–2120, August 2021.

Laganà, A., Veneziano, D., Russo, F., Pulvirenti, A., Giugno, R., Croce, C. M., and Ferro, A. Computational design of artificial RNA molecules for gene regulation. *Methods Mol. Biol.*, 1269:393–412, 2015.

Lal, A., Gunsalus, L., Nair, S., Biancalani, T., and Eraslan, G. gReLU: A comprehensive framework for DNA sequence modeling and design. *bioRxiv*, pp. 2024.09.18.613778, September 2024.

Lin, S., Hong, L., Wei, D.-Q., and Xiong, Y. Deep learning facilitates efficient optimization of antisense oligonucleotide drugs. *Mol. Ther. Nucleic Acids*, 35(2):102208, June 2024.

Linder, J. and Seelig, G. Seqprop: Stochastic sequence propagation - a keras model for optimizing DNA, RNA and protein sequences based on a predictor. https://github.com/johli/seqprop, 2019.

Linder, J. and Seelig, G. Fast activation maximization for molecular sequence design. *BMC Bioinformatics*, 22(1): 510, October 2021.

Linder, J., Bogard, N., Rosenberg, A. B., and Seelig, G. Deep exploration networks for rapid engineering of functional DNA sequences. *bioRxiv*, December 2019.

Mathews, D. H. How to benchmark RNA secondary structure prediction accuracy. *Methods*, 162-163:60–67, June 2019.

Mouchlis, V. D., Afantitis, A., Serra, A., Fratello, M., Papadiamantis, A. G., Aidinis, V., Lynch, I., Greco, D., and Melagraki, G. Advances in de novo drug design: From conventional to machine learning methods. *Int. J. Mol. Sci.*, 22(4):1676, February 2021.

Patel, A., Singhal, A., Wang, A., Pampari, A., Kasowski, M., and Kundaje, A. DART-eval: A comprehensive DNA language model evaluation benchmark on regulatory DNA. *arXiv [cs.LG]*, December 2024.

Paul, S. M., Mytelka, D. S., Dunwiddie, C. T., Persinger, C. C., Munos, B. H., Lindborg, S. R., and Schacht, A. L. How to improve R&D productivity: the pharmaceutical industry's grand challenge. *Nat. Rev. Drug Discov.*, 9(3): 203–214, March 2010.

Riley, A. T., Robson, J. M., and Green, A. A. Generative and predictive neural networks for the design of functional RNA molecules. *bioRxiv*, pp. 2023.07.14.549043, July 2023.

Sadybekov, A. V. and Katritch, V. Computational approaches streamlining drug discovery. *Nature*, 616(7958): 673–685, April 2023.

Sasse, A., Chikina, M., and Mostafavi, S. Quick and effective approximation of in silico saturation mutagenesis experiments with first-order taylor expansion. *iScience*, 27(9):110807, September 2024.

Schreiber, J. bpnet-lite: This repository hosts a minimal version of a python API for BPNet. https://github.com/jmschrei/bpnet-lite, 2020a.

Schreiber, J. Ledidi: Ledidi turns any machine learning model into a biological sequence editor, allowing you to design sequences with desired properties. https://github.com/jmschrei/ledidi, 2020b.

Schreiber, J., Lu, Y. Y., and Noble, W. S. Ledidi: Designing genomic edits that induce functional activity. *bioRxiv*, May 2020.

Shen, T., Hu, Z., Sun, S., Liu, D., Wong, F., Wang, J., Chen, J., Wang, Y., Hong, L., Xiao, J., Zheng, L., Krishnamoorthi, T., King, I., Wang, S., Yin, P., Collins, J. J., and Li, Y. Accurate RNA 3D structure prediction using a language model-based deep learning approach. *Nat. Methods*, 21 (12):2287–2298, December 2024.

Shor, J. and Cotado, S. G. Computing systems with modularized infrastructure for training generative adversarial networks, July 2023.

Sinai, S., Wang, R., Whatley, A., Slocum, S., Locane, E., and Kelsic, E. D. AdaLead: A simple and robust adaptive greedy search algorithm for sequence design. *arXiv [cs.LG]*, October 2020.

Szikszai, M., Magnus, M., Sanghi, S., Kadyan, S., Bouatta, N., and Rivas, E. RNA3DB: A structurally-dissimilar dataset split for training and benchmarking deep learning models for RNA structure prediction. *J. Mol. Biol.*, 436 (17):168552, September 2024.

Tan, X., Zhao, Z., Wang, R., and Zhu, G. Editorial: Molecular and nanoscale engineering of nucleic acid theranostics and vaccines. *Front. Bioeng. Biotechnol.*, 10:1126876, 2022.

Van Laarhoven, P. J. M. and Aarts, E. H. L. *Simulated annealing: Theory and applications*. Mathematics and Its Applications. Springer, Dordrecht, Netherlands, December 2010.

Wyss, L., Mallet, V., Karroucha, W., Borgwardt, K., and Oliver, C. A comprehensive benchmark for RNA 3D structure-function modeling. *arXiv [q-bio.BM]*, March 2025.

Xiao, T., Hong, J., and Ma, J. DNA-GAN: Learning disentangled representations from multi-attribute images. *arXiv [cs.CV]*, November 2017.

Zhang, H., Zhang, L., Lin, A., Xu, C., Li, Z., Liu, K., Liu, B., Ma, X., Zhao, F., Jiang, H., Chen, C., Shen, H., Li, H., Mathews, D. H., Zhang, Y., and Huang, L. Algorithm for optimized mRNA design improves stability and immunogenicity. *Nature*, 621(7978):396–403, September 2023.

A. Appendix

A.1. Related Work

Comparison to protein design: Computational nucleic acid design and protein design have many similarities. Both involve designing sequences of biomolecules made up of basic building blocks to achieve desired effects in a complex biological system, usually therapeutic in nature. Both sequence spaces quickly become so large as to be difficult to explore, making the same kinds of energy landscape minimization algorithms applicable to both problems. Sequences in nature from both are selected by similar kinds of evolutionary fitness constraints. Both require lab-in-the-loop feedback cycles with wetlabs to validate predictions and to identify the most valuable training data.

There are also some significant differences between designing nucleic acid sequences and proteins. The basic building blocks are different, affecting the tokenization and branching factors of computational methods. The relative importance of secondary structure as compared to tertiary structure is different between the two. In addition, the bar for clinical success is different: for example, RNA interventions often must have fewer off-target effects than biologically downstream proteins.

Comparison to nucleic acid design with structure: The models used in this benchmark take nucleotide sequences only as input. Moreover, some predictive models take additional information, such as structural information, into account. Extra information can allow models a more comprehensive understanding of their function and interactions. Examples of extra information are secondary structures like hairpins, stems, and loops, as well as tertiary interactions that determine the overall 3D configuration. However, this comes at the cost of increased computational complexity, and requires that information be present at inference time.

Comparison to generative modeling: In this work, we benchmark techniques that generate proposals by modifying input sequences and observing the effect on the property of interest. The models used to map from sequence to property are sometimes called "discriminative." Alternatively, some techniques directly generate sequences with desired properties. These are often called "generative." Some examples include Generative Adversarial Networks (Shor & Cotado, 2023; Goodfellow et al., 2014; Riley et al., 2023), such as Deep Exploration Networks (Linder et al., 2019) and DNA-GAN (Xiao et al., 2017), diffusion models (Alamdari et al., 2023), reinforcement learning (Eastman et al., 2018), and autoregressive models (Ji et al., 2021). Unlike sampling from discriminative models, these models learn to generate the distribution of "natural" sequences, and more naturally handle diversity by often having explicit tradeoffs between quality and diversity during training, generation, or both. However, training generative models often requires more data, larger compute requirements in the form of more complex training dynamics and larger hyperparameter sweeps, and a higher degree of practical machine learning experience to train.

Comparison to non-design, DNA/RNA benchmarks: A number of benchmarks exist for adjacent but distinct problems that are at the intersection of biology and machine learning. Examples of adjacent problems with existing benchmarks are: large language model on DNA (Patel et al., 2024), secondary structure prediction (Mathews, 2019; Danaee et al., 2018), inverse folding (Anderson-Lee et al., 2016), 3D structure (Becquey et al., 2021; Adamczyk et al., 2022; Szikszai et al., 2024), and function prediction (Wyss et al., 2025). In this work, we develop a benchmark for our problem: designing nucleic acid sequences to optimize specific properties.

A.2. Design tasks

A.2.1. Cell-type specific cis-regulatory activity

Malinois models (Gosai et al., 2024) are convolutional neural network models that predict cell-type-specific cis-regulatory element (CRE) effects directly from the nucleotide sequence. Each of three models predicts the CRE effect on a different cell-line: K562 (bone marrow), HepG2 (liver), and SK-N-SH (neural). The authors of (Gosai et al., 2024) aggregated data from multiple projects from a single lab. Starting sequences were random 200 nucleotide sequences flanked on either end by 200 real sequences from the MPRA dataset.

A.2.2. Transcript factor binding

Transcription factors (TFs) are proteins that regulate gene expression by binding to specific DNA sequences. The strength of this binding, known as binding affinity, directly influences whether genes are activated or repressed. Accurately predicting these affinities is important for understanding gene regulation mechanisms and how genetic variations can disrupt normal cellular function, both of which have ramifications on understanding diseases. In practical applications, improved binding

affinity predictions enable more precise genetic engineering, better interpretation of disease-associated variants in non-coding regions, more effective drug development targeting transcription factor pathways, and enhanced capabilities in synthetic biology where precise gene expression control is essential. Despite significant progress using machine learning approaches and high-throughput experimental techniques, accurately modeling the complex biophysical interactions that determine binding affinity remains an open challenge requiring integrated experimental and computational innovations.

We used models trained to predict binding affinity for 12 TF proteins from (Schreiber, 2020b). The models were trained using the BPNet-lite repository (Schreiber, 2020a), a lightweight implementation of BPNet (Avsec et al., 2021b). These are convolutional neural networks that use DNA sequence to predict base-resolution binding profiles of transcription factors. The BPNet-lite models from (Schreiber, 2020b) were trained on data from the ENCODE portal (Hitz et al., 2023) and hosted on Zenodo. Each model was trained to predict the log-count amount of binding over the given region. The data was taken from K562 cell-line data on the Encyclopedia of DNA Elements (ENCODE) consortium portal. The start sequences for this task were 3000 bp sequences mined from chromosome 1 of the human HG38 genome assembly for initial low binding affinity (see "Start Sequences" section for more details).

A.2.3. Selective gene expression

For the Enformer tasks, we used the Enformer network (Avsec et al., 2021a) to design sequences that maximized selectivity of expression by increasing expression in muscle cells while minimizing expression in neural cells. We used regular expressions to match 71 Enformer output tracks corresponding to muscle expression (maximized), and 25 output tracks corresponding to muscle suppression (minimized). In addition, we included 11 output tracks related to expression in neural cells (minimized) and 12 output tracks related to suppression in neural cells (maximized).

The Enformer network takes roughly 200K nucleotide sequences as inputs, and is a significantly larger model than other models used in NucleoBench and generally other models used for design (see Table 1). To our knowledge, this is the first time this model has been used for design. To make the problem tractable for all NucleoBench designers (especially the backpropagation ones), we restricted the number of nucleotides that could be modified to 256.

Start sequences for the Enformer task were determined by mining the human genome for sequences with low muscle expression or low selective muscle expression. Specifically, we randomly selected sequences from hg38 until we had found 100 sequences below a threshold expression or selective expression level, according to the Enformer model.

NucleoBench selected which 256 nucleotides would be modifiable based on DNASE. Within each of the 100 start sequences, we selected the 256 nucleotides with the highest amount of predicted DNASE activity, according to Enformer.

A.3. Sequence Designers

A.3.1. AdaBeam

Adaptive Beam Search (AdaBeam) is an iterative optimization technique, motivated by the improved performance of Unordered Beam Search over Ordered Beam Search, and the strong baseline performance of AdaLead. AdaBeam begins with a population of candidate sequences and their associated fitness scores. In each round, a subset of these sequences, termed "roots," are selected as parents for generating new candidates. This selection is governed by a fitness threshold, typically retaining sequences whose fitness values are above a certain percentile of the maximum observed fitness in the current population, thus focusing computational resources on the most promising individuals. From each selected root, the algorithm initiates multiple "rollouts." A rollout consists of a series of sequential mutation and evaluation steps. A child sequence is generated by mutating its parent, where the number of point mutations is determined stochastically by a specialized sampler. The sampler itself can be fixed or adaptive. The fitness of this new child sequence is then assessed.

In this study, we use the same distribution as AdaLead for the number of edits made per round, but without the ¾ factor. While AdaLead indirectly samples from this distribution (see section "Identifying Limitations in AdaLead to Motivate Novel Sequence Designers"), AdaBeam efficiently samples from this distribution directly. Specifically, we use the following:

$$\Pr[N = n] = \begin{cases} \frac{\text{Binomial}(n,l,m)}{1-(1-m)^l} & \text{if } 1 \leq n \leq l \\ 0 & \text{otherwise} \end{cases}$$

Note that $Pr[N = 0] = 0$, and that the above denominator exactly reallocates that probability mass so that the expression is

still a probability distribution (sums to 1).

The rollout from a parent sequence continues iteratively: if a newly generated child sequence exhibits a fitness greater than or equal to its immediate parent, it becomes the parent for the next step in that rollout path. This allows for an exploration trajectory in the fitness landscape. However, if a child's fitness is lower than its parent's, that branch of the rollout is terminated to prevent pursuing less fit avenues. Additionally, rollouts are constrained by a maximum permissible length to ensure finite exploration. All unique sequences generated across all rollouts from all roots, along with their fitness scores, are collected. From this aggregated pool of candidates, the algorithm then selects the top-performing sequences, dictated by a parameter, to form the population for the subsequent iteration. This cyclical process of parent selection, adaptive mutation via rollouts, fitness evaluation, and beam-based selection drives the population towards higher fitness regions over successive generations, effectively implementing an adaptive Directed Evolution strategy for sequence optimization.

A.3.2. AdaLead

As discussed in the "Methods" section "Identifying Limitations in AdaLead to Motivate Novel Sequence Designers", we hypothesized that removing certain components of AdaLead would improve performance. Specifically, for performing edits, we replaced a repeated O(n) computation with a single O(1) computation (sampling directly from the induced distribution over number of mutations), and removed a buffer of all sequences visited and a check on repeating them. These two changes together produced strictly better results (Friedman test stat = 83, $p < 2.5 \times 10^{-16}$, one-sided t-test to establish improvement: t=4.5, $p < 3.2 \times 10^{-6}$). Second, we note that AdaLead without recombination always outperformed AdaLead with recombination (Supplemental Per-Task Results. This result was also found in (Gosai et al., 2024)).

Gradient masking, in PyTorch, multiplies the input by a constant-Tensor mask of 0s and 1s. It is implemented as follows:

```
idxs # a list of positions to have gradients computed
x    # input tensor
gradient_mask = torch.zeros_like(x)
gradient_mask[idxs] = 1
x = x * gradient_mask
y = large_network(x)
```

Listing 1. Gradient masking technique.

In contrast, the memory-efficient "gradient concatenation" constructs the input Tensor from two sub-Tensors. One is marked for gradients, the other is not. It is more memory efficient when the backwards pass is much more memory intensive than the forward pass. It is implemented as follows:

```
idxs # a list of positions to have gradients computed
x  # input tensor

# Form the basis of the gradient-free pieces of the input.
no_gradient = x.clone().detach()
no_gradient.requires_grad = False

# Form the basis of the gradient pieces of the input.
x_grad = x[idxs].clone().detach()
x_grad.requires_grad = True
x_grad_i = {idx: i for i, idx in enumerate(idxs)}

# Combine slices from the correct source, according to whether or not gradients are
required.
tensor_slices = [get_gradient_slice(i) if i in idxs else get_no_gradient_slice(i) for i
in range(x.shape)]
x = torch.concat(tensor_slices, dim=2)

y = large_network(x)
# Compute gradient of y with respect to x
```

Listing 2. Gradient concatenation. This memory-efficient technique enables backpropagation through Enformer.

Gradient concatenation enables our hybrid and novel gradient algorithms to design sequences on Enformer tasks, while

Ledidi and FastSeqProp cannot (Table 5). For more details, see the "apply_gradient_mask" implementation in NucleoBench.

A.3.3. GRADIENT EVO

We were able to improve the worst-performing algorithm (Directed Evolution) into one of the top performing optimizers by leveraging gradient information to select the location of edits. One important question, though, is which steps are most important to have gradient guidance. Since we decomposed each step in this designer into two phases, an edit location selection phase and an edit phase, we were able to quantify the relative importance of gradient guidance.

A paired t-test across all the best hyperparameters for each task did not show a statistical difference between the performances of location-only Gradient Evo and the version of Gradient Evo that used gradients to select the nucleotide edit (p=0.98, n=1900). This suggests that it is more important to apply gradient guidance to selecting *where* to make edits than it is to apply to *what* the edits should be.

A.4. Per-task performance

Table A.1. 95% confidence intervals for the best per-task results. For each (task, algorithm, hyperparameters), the 95% confidence interval was computed across start sequences and random seeds using the final optimized energies (lower is better). Red is the best, blue second best, per column.

Algorithm	Malinois			BPNet												Enformer
	K562	HepG2	SK-N-S	ATAC	CTCF	ELF3	ELF4	GATA2	JUNB	MAX	MECOM	MYC	OTX1	RAD21	SOX6	↑muscle ↓liver
Directed Evolution	**-15.4, -15.3**	-14.5, -14.2	-14.2, -13.8	-2.2, -1.9	-0.9, -0.7	-1.4, -1.3	0.6, 0.7	-0.9, -0.7	-4.1, -4.1	-0.2, -0.0	-1.6, -1.3	-0.1, 0.0	-4.5, -4.4	-2.1, -1.9	-3.4, -3.3	-10319, -5750
Simulated Annealing	**-15.4, -15.4**	-10.4, -10.3	-8.2, -8.2	-3.8, -3.8	-3.7, -3.7	-2.8, -2.8	-2.3, -2.3	-3.0, -2.9	-4.5, -4.5	-2.7, -2.7	-2.5, -2.4	-1.4, -1.4	-4.8, -4.8	-4.0, -4.0	-4.1, -4.1	-12682, -6788
AdaLead	**-15.4, -15.4**	**-15.4, -15.4**	**-15.4, -15.4**	**-38.3, -37.2**	-14.9, -14.7	-25.2, -24.9	-18.5, -18.2	-19.4, -18.9	**-17.3, -17.3**	-17.4, -17.2	-36.7, -36.1	-15.7, -15.3	**-5.9, -5.9**	-17.3, -16.8	-31.7, -31.1	-18919, -11011
FastSeqProp	**-15.4, -15.4**	**-15.4, -15.3**	**-15.2, -15.1**	-19.3, -18.6	-12.2, -12.1	-25.1, -25.0	-18.7, -18.5	-21.7, -21.6	-15.6, -15.4	-16.3, -16.1	-34.7, -34.4	-13.0, -12.9	-5.0, -5.0	-16.8, -16.6	-29.6, -29.5	**-72696, -65458**
Ledidi	**-15.4, -15.4**	**-15.4, -15.4**	**-15.4, -15.3**	**-46.8, -45.8**	**-18.2, -17.7**	**-28.6, -28.4**	**-20.8, -20.5**	**-27.0, -26.4**	-15.7, -15.4	**-20.6, -20.3**	**-40.3, -39.7**	**-16.4, -15.9**	-4.6, -4.6	**-25.3, -24.5**	**-33.0, -32.7**	**-1186192, -1040939**
Ordered Beam	**-15.4, -15.4**	**-15.4, -15.4**	**-15.4, -15.3**	-2.4, -2.1	-6.1, -6.0	-10.0, -9.9	-6.9, -6.8	-8.6, -8.4	-7.3, -7.2	-7.5, -7.3	-11.8, -11.6	-6.5, -6.3	-5.3, -5.2	-7.5, -7.3	-12.0, -11.8	-14972, -8515
Unordered Beam	**-15.4, -15.4**	**-15.4, -15.4**	**-15.4, -15.4**	-31.4, -30.8	-14.0, -13.6	-22.7, -22.4	-17.0, -16.9	-16.8, -16.6	-16.9, -16.8	-14.9, -14.6	-33.0, -32.4	-13.8, -13.4	**-5.8, -5.8**	-15.1, -14.2	-29.3, -28.9	-21869, -12796
Gradient Evo	**-15.4, -15.4**	**-15.2, -14.9**	-14.6, -13.9	-35.7, -34.7	-14.4, -14.0	-23.1, -22.8	-17.1, -16.7	-17.1, -16.8	-17.0, -16.9	-15.3, -15.0	-33.3, -32.6	-13.7, -13.3	**-5.8, -5.8**	-15.4, -14.5	-29.8, -29.2	-11130, -5962
AdaBeam	**-15.4, -15.4**	**-15.4, -15.4**	**-15.4, -15.4**	**-46.9, -46.1**	**-17.2, -16.9**	**-32.1, -32.0**	**-23.4, -23.0**	**-25.7, -25.1**	**-17.6, -17.5**	**-19.7, -19.4**	**-43.7, -43.3**	**-22.9, -22.4**	**-5.9, -5.9**	**-22.5, -21.5**	**-36.4, -36.0**	-23894, -13523

A.5. Per-task variability due to random seed

Table A.2. Performance variability based on random seed. For each (task, algorithm, hyperparameters), we show the (25th percentile, 75th percentile) for the best performing hyperparameters.

Algorithm	Malinois			BPNet												Enformer
	K562	HepG2	SK-N-S	ATAC	CTCF	ELF3	ELF4	GATA2	JUNB	MAX	MECOM	MYC	OTX1	RAD21	SOX6	↑muscle ↓liver
Directed Evolution	(0.00, 0.00)	(0.95, 1.97)	(1.32, 2.35)	(0.36, 0.73)	(0.35, 0.78)	(0.12, 0.29)	(0.15, 0.42)	(0.41, 0.72)	(0.06, 0.13)	(0.28, 0.62)	(0.11, 0.32)	(0.30, 0.55)	(0.01, 0.03)	(0.36, 0.72)	(0.09, 0.21)	(646.21, 3273.90)
Simulated Annealing	(0.00, 0.00)	(0.26, 0.49)	(0.18, 0.28)	(0.06, 0.10)	(0.08, 0.12)	(0.07, 0.12)	(0.04, 0.06)	(0.08, 0.15)	(0.01, 0.02)	(0.07, 0.11)	(0.05, 0.09)	(0.06, 0.10)	(0.00, 0.01)	(0.09, 0.14)	(0.05, 0.08)	(694.23, 6233.92)
AdaLead	(0.00, 0.00)	(0.00, 0.00)	(0.00, 0.05)	(1.38, 2.11)	(0.44, 0.75)	(0.45, 0.77)	(0.42, 0.76)	(0.56, 0.87)	(0.10, 0.15)	(0.33, 0.58)	(1.00, 1.63)	(0.59, 0.88)	(0.01, 0.01)	(0.56, 1.07)	(0.87, 1.45)	(1051.72, 7289.75)
FastSeqProp	(0.00, 0.00)	(0.00, 0.20)	(0.41, 0.90)	(2.47, 3.79)	(0.20, 0.32)	(0.38, 0.60)	(0.38, 0.57)	(0.49, 0.77)	(0.22, 0.36)	(0.45, 0.69)	(0.97, 1.62)	(0.54, 0.86)	(0.01, 0.02)	(0.73, 1.08)	(0.55, 0.87)	(15306.33, 25015.24)
Ledidi	(0.00, 0.00)	(0.00, 0.01)	(0.00, 0.02)	(1.53, 2.43)	(0.56, 1.09)	(0.42, 0.68)	(0.43, 0.67)	(0.70, 1.15)	(0.45, 0.74)	(0.35, 0.67)	(0.93, 1.53)	(0.67, 1.07)	(0.01, 0.02)	(0.90, 1.48)	(0.46, 0.82)	(120364.95, 243801.88)
Ordered Beam	(0.00, 0.00)	(1.50, 1.88)	(1.86, 2.24)	(0.29, 0.52)	(0.36, 0.54)	(3.97, 4.74)	(2.69, 3.24)	(1.11, 1.91)	(1.62, 2.23)	(0.94, 1.53)	(7.15, 8.38)	(2.14, 3.52)	(0.03, 0.05)	(1.32, 2.02)	(4.70, 5.63)	(778.27, 6819.78)
Unordered Beam	(0.00, 0.00)	(0.00, 1.54)	(1.05, 2.19)	(1.73, 2.77)	(0.50, 0.82)	(0.47, 0.85)	(0.46, 0.71)	(0.57, 0.83)	(0.12, 0.19)	(0.50, 0.75)	(1.11, 1.66)	(0.55, 0.91)	(0.01, 0.01)	(0.65, 1.71)	(1.03, 1.66)	(931.13, 7811.52)
Gradient Evo	(0.00, 0.00)	(1.14, 1.86)	(1.66, 2.51)	(1.63, 2.62)	(0.53, 0.83)	(0.60, 0.82)	(0.54, 0.75)	(0.52, 0.81)	(0.14, 0.20)	(0.44, 0.72)	(1.26, 1.88)	(0.60, 0.87)	(0.01, 0.01)	(0.74, 1.35)	(1.13, 1.77)	(623.83, 2415.36)
AdaBeam	(0.00, 0.00)	(0.00, 0.04)	(0.54, 1.08)	(1.29, 1.91)	(0.44, 0.71)	(0.35, 0.53)	(0.39, 0.70)	(0.96, 1.47)	(0.08, 0.14)	(0.32, 0.57)	(0.74, 1.18)	(0.58, 0.99)	(0.01, 0.01)	(0.84, 1.40)	(0.64, 1.01)	(1157.73, 6996.59)

A.6. Per-task variability due to start sequence

Table A.3. Performance variability based on start sequences. For each (task, algorithm, hyperparameters), We show the (mean +- standard deviation) for the best performing hyperparameters.

Algorithm	Malinois			BPNet												Enformer
	K562	HepG2	SK-N-S	ATAC	CTCF	ELF3	ELF4	GATA2	JUNB	MAX	MECOM	MYC	OTX1	RAD21	SOX6	↑muscle ↓liver
Directed Evolution	-15.32 ± 0.19	-14.36 ± 0.85	-13.99 ± 0.89	-2.02 ± 0.66	-0.83 ± 0.51	-1.37 ± 0.19	0.65 ± 0.43	-0.79 ± 0.39	-4.07 ± 0.07	-0.13 ± 0.53	-1.44 ± 0.57	-0.04 ± 0.28	-4.44 ± 0.04	-1.99 ± 0.57	-3.34 ± 0.11	-8035.22 ± 11514.42
Simulated Annealing	-15.39 ± 0.00	-10.33 ± 0.19	-8.18 ± 0.11	-3.82 ± 0.04	-3.66 ± 0.05	-2.78 ± 0.05	-2.30 ± 0.02	-2.96 ± 0.06	-4.49 ± 0.01	-2.69 ± 0.04	-2.45 ± 0.03	-1.38 ± 0.04	-4.77 ± 0.00	-4.03 ± 0.05	-4.12 ± 0.03	-9735.61 ± 14852.95
AdaLead	-15.39 ± 0.00	-15.39 ± 0.00	-15.39 ± 0.00	-37.74 ± 2.58	-14.79 ± 0.66	-25.03 ± 0.71	-18.35 ± 0.84	-19.12 ± 1.15	-17.30 ± 0.13	-17.29 ± 0.61	-36.43 ± 1.46	-15.48 ± 0.86	-5.87 ± 0.01	-17.04 ± 1.39	-31.42 ± 1.50	-14965.50 ± 19927.95
FastSeqProp	-15.39 ± 0.00	-15.33 ± 0.31	-15.15 ± 0.48	-18.94 ± 1.99	-12.18 ± 0.28	-25.06 ± 0.29	-18.59 ± 0.63	-21.67 ± 0.37	-15.50 ± 0.38	-16.21 ± 0.59	-34.54 ± 0.83	-12.95 ± 0.47	-5.00 ± 0.04	-16.75 ± 0.50	-29.55 ± 0.41	-69077.62 ± 18239.27
Ledidi	-15.39 ± 0.00	-15.38 ± 0.02	-15.36 ± 0.07	-46.32 ± 2.55	-17.93 ± 1.16	-28.50 ± 0.57	-20.64 ± 0.72	-26.68 ± 1.60	-15.55 ± 0.87	-20.47 ± 0.67	-40.01 ± 1.51	-16.14 ± 1.22	-4.64 ± 0.03	-24.87 ± 2.02	-32.84 ± 0.74	-1113565.75 ± 366020.88
Ordered Beam	-15.39 ± 0.00	-15.39 ± 0.00	-15.34 ± 0.26	-2.22 ± 0.67	-6.01 ± 0.25	-9.93 ± 0.36	-6.88 ± 0.26	-8.52 ± 0.41	-7.21 ± 0.27	-7.39 ± 0.43	-11.72 ± 0.59	-6.39 ± 0.48	-5.25 ± 0.02	-7.38 ± 0.61	-11.93 ± 0.57	-11743.73 ± 16270.57
Unordered Beam	-15.39 ± 0.00	-15.39 ± 0.00	-15.39 ± 0.00	-31.11 ± 1.66	-13.79 ± 0.87	-22.53 ± 0.83	-16.95 ± 0.47	-16.71 ± 0.49	-16.85 ± 0.17	-14.79 ± 0.79	-32.71 ± 1.67	-13.62 ± 0.96	-5.83 ± 0.01	-14.62 ± 2.22	-29.13 ± 0.92	-17332.86 ± 22861.82
Gradient Evo	-15.39 ± 0.00	-15.06 ± 0.96	-14.26 ± 1.95	-35.17 ± 2.49	-14.19 ± 0.87	-22.96 ± 0.75	-16.89 ± 0.84	-16.98 ± 0.82	-16.96 ± 0.17	-15.13 ± 0.85	-32.91 ± 1.72	-13.47 ± 0.90	-5.84 ± 0.01	-14.95 ± 2.50	-29.48 ± 1.48	-8546.06 ± 13022.21
AdaBeam	-15.39 ± 0.00	-15.39 ± 0.00	-15.39 ± 0.00	-40.95 ± 8.24	-16.96 ± 0.77	-31.74 ± 0.54	-23.03 ± 0.66	-24.92 ± 1.73	-17.53 ± 0.13	-18.21 ± 1.19	-41.19 ± 1.15	-20.92 ± 1.00	-5.89 ± 0.01	-21.24 ± 2.80	-35.97 ± 1.09	-18708.53 ± 26135.83