Liu *et al. BMC Bioinformatics* (2026)27:253
https://doi.org/10.1186/s12859-026-06696-6

BMC Bioinformatics

SOFTWARE

Open Access

PoolParty: streamlined design of DNA sequence libraries in Python

Zhihan Liu[1], Aidan Cordero[1] and Justin B. Kinney[1*]

*Correspondence:
Justin B. Kinney
jkinney@cshl.edu
[1]Simons Center for Quantitative Biology, Cold Spring Harbor Laboratory, 1 Bungtown Rd., Cold Spring Harbor, New York 11724, USA

Abstract

Background Computationally designed DNA sequence libraries are essential components of massively parallel reporter assays (MPRAs), deep mutational scanning (DMS) experiments, and other multiplex assays of variant effect (MAVEs). They are also increasingly used in silico to analyze genomic AI models. Designing these libraries, however, remains tedious and error-prone due to the scarcity of purpose-built software.

Results Here we describe PoolParty, a Python package that streamlines the design of complex oligo pools using a simple but flexible API. In PoolParty, each library is represented by a computational graph that can be specified in just a few lines of code. Over 50 built-in operations cover nucleotide- and codon-level mutagenesis, motif insertion, barcode generation, and more. PoolParty automatically generates informative names for each sequence and provides "design cards" detailing how each sequence was generated. Visualization methods let users quickly audit library content and inspect the underlying graph. PoolParty thus transforms oligo pool design from a tedious task requiring custom functions and scripts into a structured, transparent, and reproducible process.

Conclusions PoolParty streamlines the design of DMS, MPRA, and other multiplex assay libraries, and the design cards it provides can help researchers systematically probe and interpret genomic AI models. PoolParty can also be extended to support new assays and analysis strategies as they emerge.

Keywords Sequence design, Library design, Massively parallel reporter assay, Deep mutational scanning, Multiplex assay of variant effect, Genomic AI, In silico experiments, Python software, Directed acyclic graph

Background

Designed libraries of DNA sequences (oligonucleotide pools) have become essential tools in functional genomics. In massively parallel reporter assays (MPRAs), complex libraries of DNA sequences are routinely used to measure the regulatory activities of tens of thousands of candidate regulatory elements or their variants in a single experiment [1–13]. In deep mutational scanning (DMS) experiments, coding sequences that have been either systematically or randomly mutagenized are commonly used to map

© The Author(s) 2026. **Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

protein fitness landscapes [14–20]. Oligo pools can also be used in high-throughput biochemical assays, such as measurements of transcription factor specificity [21–27]. These experiments are all examples of multiplex assays of variant effect (MAVEs), a large and expanding family of high-throughput methods that quantitatively link sequence to function at scale [28–30]. Beyond the wet lab, designed sequence libraries are increasingly used in silico to probe the behavior of genomic AI models [31–35] and to augment genomic AI training data [36].

Computationally designing these libraries is straightforward in principle, but in practice it can be tedious and error-prone. Typical applications require oligo pools comprising multiple types of variant sequences. A DMS library, for example, might combine exhaustive single-amino-acid substitutions, sampled higher-order mutants, multiple copies of wild-type and control sequences, and barcodes—each generated by a different procedure and subject to different coverage requirements. An MPRA library might tile multiple transcription factor binding sites across a regulatory element in all possible permutations and orientations, then assign each variant multiple barcodes. Coordinating this kind of combinatorial logic in a one-off script quickly becomes unwieldy, and often produces errors that are hard to catch.

Existing tools can help with parts of this process. General-purpose sequence toolkits [37–39] can manipulate sequences but have no notion of a variant library as a structured object. Consequently, users must write custom scripts to handle the combinatorial logic themselves. Several other tools automate library design or sequence perturbation for particular applications. For example, VaLiAnT [40] generates mutational scanning libraries of both coding and noncoding regions; supported mutation types include single-nucleotide substitutions, amino acid substitutions, and in-frame deletions. For MPRAs, MPRAnator [41] provides modules for motif placement, SNP design, and negative control generation (see also [42]). For in silico experiments, tangermeme [43] supports sequence perturbations and the analysis of genomic deep learning models. However, each of these tools is targeted to specific applications and, as a result, no individual tool can generate the full range of library designs in common use. For example, VaLiAnT does not support regulatory sequence libraries containing random combinations of transcription factor binding sites, while MPRAnator and tangermeme do not support codon-aware mutagenesis of protein-coding sequences.

Here we present PoolParty, a Python package that enables the streamlined design of complex DNA sequence libraries. In PoolParty, libraries are represented as objects called *Pools*, while individual steps in the sequence generation process are called *Operations*. Using a concise and transparent syntax, users chain together Operations into a directed acyclic graph (DAG)—an approach inspired by deep learning frameworks—that yields the desired Pool. The DAG describes the high-level logic used to generate sequences, while the procedural details and bookkeeping are handled internally. Importantly, no sequences are generated until the user requests them, and it is simple to generate small test sets of sequences. This allows users to explore multiple design options without having to generate complete libraries at each iteration. When sequences are generated, each is automatically assigned an informative name and paired with a *design card* that records the specific procedures used to construct it. We demonstrate PoolParty through three example applications: a DMS library for protein GB1, an MPRA library probing transcriptional regulatory grammar, and an in silico experiment characterizing the genomic

deep learning model SpliceAI [44], in which design cards provide the covariates for surrogate modeling.

Implementation

Pools and Operations

PoolParty is built around two core abstractions: Pools and Operations. Pools represent collections of sequences—either the final library that the user wishes to generate, or intermediate sets of sequences out of which the final library is constructed. Operations represent the steps used to create Pools. When designing a library, users chain together multiple Operations into a DAG that yields the desired Pool, called the *root Pool*. Sequences are then generated on demand from the root Pool. By delaying sequence generation until it is explicitly requested, users can test many different library designs before committing to generating the full library. For example, users can inspect a few sampled sequences from each one of multiple proposed designs.

PoolParty provides over 50 built-in Operations that users can choose from when constructing DAGs. Each Operation takes zero or more Pools as input and returns exactly one Pool as output. It is useful to think of Operations as falling into four functional categories (Fig. 1A): source, transformation, composition, and state Operations. *Source Operations* create the initial Pools from which all downstream Pools are constructed. The Pools they create may represent user-specified sequences loaded directly from

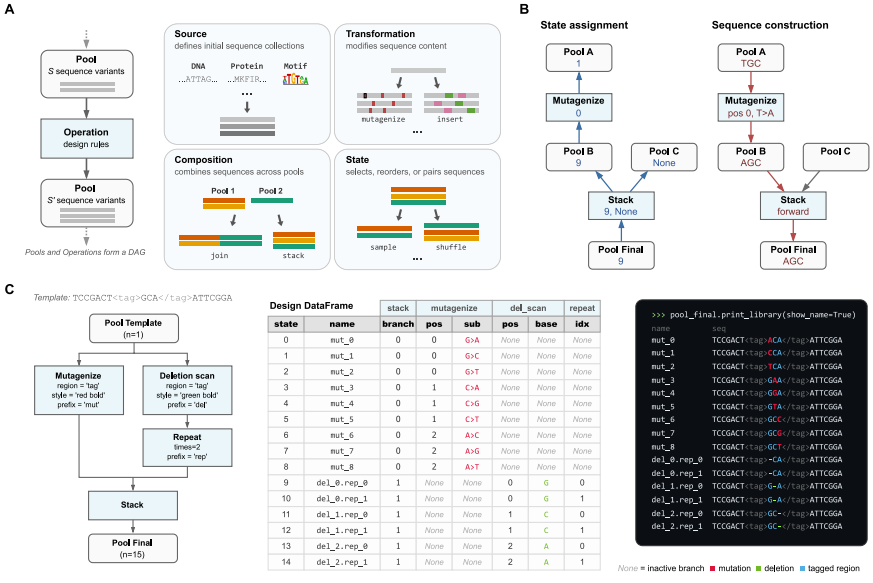


Fig. 1 PoolParty builds libraries from Pools and Operations. **A** Pools represent collections of sequences, and Operations generate new Pools based on their inputs (left). Users chain multiple Operations into a directed acyclic graph (DAG) that specifies the desired library. Operations roughly fall into four functional categories: source, transformation, composition, and state (right). **B** Sequence generation proceeds in two steps: *state assignment* and *sequence construction*. For each desired sequence, PoolParty passes to the root Pool a number (called the *state*) that specifies which sequence to generate. During state assignment (left), this state is propagated backward through the DAG, resolving the internal state of every Operation along the way. During sequence construction (right), each Operation generates and combines sequence fragments according to its assigned state, producing the final sequence at the root Pool. When multiple Pools are combined by `stack`, only one parent is active for a given state; inactive branches receive no state (`None`) and are shown as gray arrows. This example is expanded on in Supplemental Information, Fig. S1. **C** Example workflow. The template sequence contains a user-specified region (called "tag") that is targeted for both mutagenesis and deletion scanning. Left: the DAG specifying the sequence library. Center: the design card DataFrame returned by the DAG. Right: styled sequence output, with colors indicating mutations (red), deletions (green), and the tag region (blue)

FASTA or VCF files, DNA sequences generated using position weight matrices (PWMs) or IUPAC motifs, random k-mers, barcodes, and so on. *Transformation Operations* modify sequence content, e.g., generating sequence variants through point mutations, insertions, deletions, recombination, or shuffling. ORF-aware Operations are provided for transformations that operate on open reading frames. *Composition Operations* combine sequences from multiple parent Pools, either by concatenating parent sequences end-to-end (the `join` Operation) or by merging multiple Pools into one Pool (the `stack` Operation). *State Operations* affect which sequences are kept in a Pool and how they are ordered, and include methods to select, reorder, filter, and replicate sequences. Sequences can be filtered based on sequence length, GC content, homopolymer length, sequence complexity, restriction enzyme recognition sites, or other user-defined criteria.

Sequence generation

To generate a sequence library, PoolParty iterates through a series of *states*. Each state is an integer that, together with a random seed, uniquely determines a sequence. The generation of each individual sequence proceeds in two steps: *state assignment* and *sequence construction* (Fig. 1B; Supplemental Information, Fig. S1). This process begins when the corresponding state is passed to the root Pool. State assignment involves a *backward pass* through the DAG, from the root Pool toward the source Pools. Each Pool passes the state it receives to the Operation that created it. Each Operation then decomposes this state into an *internal state* (which it keeps) and zero or more *component states*, each of which is passed to a distinct input Pool of that Operation. Sequence construction then occurs during a *forward pass* through the DAG, from the initial Pools toward the root Pool: each Operation constructs a sequence based on its internal state and the sequences returned by its input Pools, then passes this sequence to the Pool it creates. The final constructed sequence is then received from the root Pool and recorded.

Operation modes

The specific way that each Operation interprets its internal state is determined by its *mode*.

- In *sequential mode*, the Operation exhaustively enumerates a finite set of sequence designs. For example, when `mutagenize` is instructed to generate single-nucleotide mutations in sequential mode, it yields one variant for each possible character change at each target position. Its internal state specifies which mutation to apply at which position.
- In *random mode*, the Operation samples sequence designs stochastically. For example, when `mutagenize` operates in random mode, it randomly selects which positions to mutate and which mutations to introduce. Its internal state, combined with a master seed that determines the specific random draws, ensures reproducibility.
- In *fixed mode*, there is no internal state. Rather, output sequences are uniquely determined by the input sequences. One example of a fixed-mode operation is `join`, which concatenates its input sequences without modification.

StateTracker

The way each Operation decomposes the state it receives during the backward pass is designed to reverse the way it constructs sequences during the forward pass. For most Operations, the set of possible output states is the Cartesian product of the possible internal states of the Operation together with the possible states of each input Pool. In such cases, the Operation uses mixed-radix decomposition to split the received state into its component states. The `stack` Operation is different: its output states are the disjoint union of its input Pool states. The `stack` Operation therefore uses the state it receives to select which input Pool to use, and to instruct that Pool (but not the other Pools) which sequence to return. Keeping track of these rules across a complex DAG requires specialized bookkeeping. We therefore developed a back-end package called StateTracker that automates state composition and decomposition during forward and backward passes through the DAG.

Sequence regions

One may want to perform different Operations on different parts of a sequence. To facilitate this, PoolParty lets users mark regions of interest within sequences using XML-style tags. PoolParty supports both opening/closing tag pairs (e.g., `AAA<cre>CCCGGG</cre>TTT`) and self-closing tags (e.g., `ACGT<ins/>ACGT`). Operations can then be instructed to act only on a specified region, as illustrated in Fig. 1C. Tags persist through the DAG and remain valid even when upstream Operations change the content of a region. Multiple Operations can thus target modifications to the same region in series.

Sequence metadata

It is often useful to record how each sequence within a library was constructed. PoolParty provides three complementary ways to do this (Fig. 1C).

- *Design cards* record the specific changes applied to each sequence by each Operation. Users specify which Operations and Operation-specific variables to track, including user-defined sequence scores computed with the `score` Operation. As each sequence is generated, the root Pool returns it together with a design card that records the tracked variable values for that sequence. PoolParty compiles these into a DataFrame in which each row contains a generated sequence and its design card.
- *Sequence names* summarize at a glance how each sequence was constructed. Users assign a prefix to each Operation they wish to track. PoolParty then pairs each prefix with the corresponding Operation's internal state to form a token (e.g., `mut_03`), and concatenates these tokens to form the sequence's name. For example, `wt.mut_03.bc_01` might denote a wild-type sequence with mutation variant number 3 and barcode number 1.
- *Sequence styling* allows sequences to be annotated with colors and formatted text that reflect each one's construction history. For example, `mutagenize` can be instructed to mark the positions it mutates in bold yellow text, making these positions stand out from the rest of the sequence. This formatting is accomplished using ANSI escape codes, so that output renders correctly in command-line terminals, Jupyter notebooks, and other environments that support basic text formatting. These styles combine in a natural manner as sequences propagate through the DAG. For example,

a mutated position can be underlined even if it occurs within a sequence region that is already colored blue.

PoolParty also provides a `stats` method for summarizing a DNA Pool, including its unique and duplicate sequence counts, pairwise Hamming distances, and homopolymer frequencies.

Implementation and extensibility

PoolParty requires Python 3.10 or later, is designed to have minimal dependencies, and is distributed via PyPI. Documentation, tutorials, and example notebooks are available on the ReadTheDocs and GitHub project websites. PoolParty is also extensible: users can create new Operation types by subclassing the Operation base class and implementing a few required methods. Custom Operations automatically inherit support for state tracking, design card generation, and visualization, allowing researchers to add domain-specific functionality without reimplementing core infrastructure.

SpliceAI surrogate modeling analysis

This section provides technical details for the surrogate modeling analysis presented in Results. Using the human 5' splice site (5'ss) position weight matrix [45], we sampled 9-mers representing cryptic 5'ss sequences and scored them with MaxEntScan [46]. Sequences were binned into 100 quantiles of MaxEntScan score (hereafter, strength), and 20 sequences were drawn from each bin (2,000 total). Each 9-mer was inserted at 100 positions flanking the canonical 5'ss of SMN2 exon 7 and paired with a matched control in which the GT dinucleotide was disrupted (T > A at + 2).

SpliceAI [44] is a genomic deep learning model for splice-site prediction in humans. We used the published five-model ensemble to predict the canonical 5'ss probability using ± 5 kb of genomic context (GRCh38). For each library-control pair, we computed the difference in logit-transformed predictions at the canonical 5'ss. These per-sequence values were averaged within each cell of the resulting 100 × 100 position-by-strength grid.

We fit generalized additive models (GAMs) to the cell means using pyGAM [47], with smooth terms for position, strength, and their interaction. Models were weighted by the inverse variance of each cell mean. Exonic and intronic regions were modeled separately. Model performance was evaluated by checkerboard cross-validation of the grid.

Use of large language models

The authors used large language models (primarily Claude Opus 4.5 and 4.6, Anthropic) to assist with code development, debugging, and documentation writing. The manuscript was written entirely by the authors, with LLM assistance only at the copy-editing stage. The figures were also prepared by the authors; LLMs were used only to help write the scripts that generate the plots. The authors affirm that they are fully responsible for the content of the manuscript, the codebase, and the documentation.

Results

Example 1: DMS library for protein GB1

As a first example, we use PoolParty to reconstruct and extend the library used in a DMS experiment on protein GB1 by Olson et al. [17]. Their library contained the wild-type

GB1 coding sequence, all single-amino-acid substitutions (excluding the start codon), and all possible pairs of amino acid substitutions. This library is substantially larger than most DMS libraries, containing over half a million variants, and has become a benchmark dataset in the quantitative modeling of protein sequence-function relationships [48–51]. To the Olson et al. library we added random higher-order amino acid substitutions as well as 100 replicates of the wild-type sequence.

Fig. 2 A illustrates the DAG used to construct the resulting library; the corresponding Python code is shown in Fig. 2B. First, we use the `from_seq` Operation to define a Pool named `orf_pool` that represents the wild-type GB1 coding sequence. This Pool is then passed to four separate Operations that define the four library components. The first is a `repeat` Operation that produces 100 copies of the wild-type sequence. The second and third are `mutagenize_orf` Operations running in sequential mode: one generates all 1,045 single-amino-acid substitutions, and the other generates all 536,085 pairwise amino acid substitutions. The fourth is a `mutagenize_orf` Operation running in random mode, which mutates each amino acid position with 10% probability to generate 10,000 higher-order variants. We note that `mutagenize_orf` supports multiple codon mutation types; this example uses `missense_only_first`, which selects the codon most frequently used in the human genome for each target amino acid. These four Pools are then merged by `stack` into the root Pool, yielding a library of 547,230 sequences in total. The design card for each variant (Fig. 2C) reports which component it belongs to, the specific mutations it contains, and the replicate index (for wild-type copies). Users



Fig. 2 A deep mutational scanning (DMS) library for protein GB1. **A** DAG and representative sequences for each of the four library components: replicates of the wild-type sequence, all single-amino-acid substitutions, all pairwise amino acid substitutions, and random higher-order mutants. Mutated codons are highlighted with amino acid translations shown beneath. **B** Python code implementing the DAG. **C** Design cards reporting the library component, mutation positions, and amino acid changes for representative variants. **D** DAG structure displayed by `print_dag()`

can also inspect the DAG structure by calling the `print_dag` method of the root Pool (Fig. 2D).

Example 2: MPRA library for regulatory grammar

MPRA libraries used to study regulatory grammar often vary the positions and orientations of transcription factor binding sites (TFBSs) [3, 5, 52–56]. For example, Georgakopoulos-Soares et al. [56] studied the impact of order and orientation on gene expression by testing over 200,000 sequences containing all pairwise and triplet arrangements of 18 TFBSs. Designing such libraries requires substantially more complex combinatorics than standard DMS libraries do. PoolParty handles this complexity without making additional demands on the user.

As a second example application of PoolParty, we designed an MPRA library representing various arrangements of binding sites for three liver-enriched transcription factors—HNF4A, PPARA, and XBP1 (Fig. 3A, B). Following the oligo layout of Melnikov et al. [3], we start by creating a Pool representing a template sequence that contains a putatively inert 100 bp cis-regulatory element (CRE) region (`<cre>`) and a barcode region (`<bc>`). Next, we create a Pool representing the binding site for HNF4A. This Pool is then passed to the `flip` Operation, which yields both orientations of the site (forward and reverse complement). Pools for PPARA and XBP1 are created similarly. The template Pool and three TFBS Pools are then passed to the `insertion_multiscan` Operation, which samples 1,000 random placements of the TFBSs within the `<cre>` region without overlap. Because `flip` runs in sequential mode, all $2^3 = 8$ orientation combinations are generated for each placement, yielding 8,000 unique CRE variants. Each sequence is

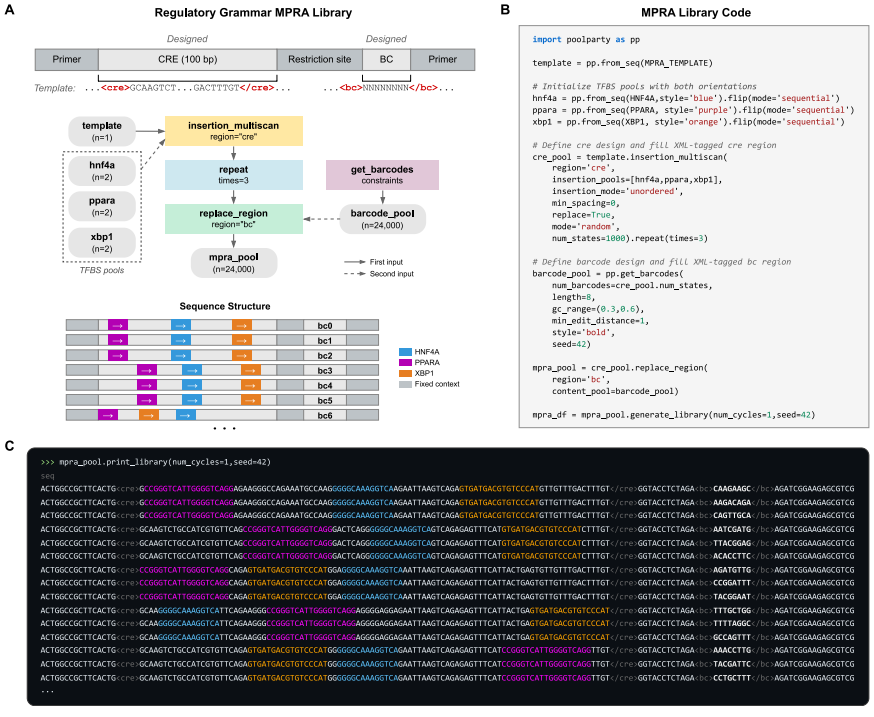


Fig. 3 A massively parallel reporter assay (MPRA) library for probing regulatory grammar. **A** Library schematic showing TFBS placements across a 100-bp CRE in representative variants, each linked to three different barcodes. **B** Python code for library construction. TFBS Pools are initialized with forward sequences and augmented with the `flip` Operation to include both orientations. **C** Output of `print_library`. TFBSs are colored by identity (HNF4A blue, PPARA purple, XBP1 orange); barcodes are shown in bold

then replicated three times using the `repeat` Operation. Finally, the `get_barcodes` Operation is used to generate unique barcode sequences that are inserted into the `<bc>` region of the template. The result is a library of 24,000 sequences (Fig. 3A).

The text styling capabilities of PoolParty help users verify the architecture of the resulting sequences. After the template Pool is created, the `stylize` Operation is used to shade the `<cre>` region gray. Each TFBS Pool is also assigned a distinct color when it is created (HNF4A blue, PPARA purple, XBP1 orange); this is specified by the `style` parameter passed to `from_seq`. Barcodes are similarly set to render in bold text (Fig. 3B). These styles compose as sequences pass through the DAG, so that when `print_library` is called (Fig. 3C), the position, order, and orientation of each binding site are immediately visible.

Example 3: surrogate modeling of SpliceAI predictions

Designed sequence libraries are also used to study genomic AI models [31–33, 35]. First, an in silico experiment is carried out in which the AI model scores all sequences in a library. The results are then analyzed to characterize model behavior. One useful way of analyzing these synthetic data is *surrogate modeling*, in which a mathematically simple model—one whose structure and parameters are directly interpretable—is fit to the predictions.

In addition to streamlining the design of sequence libraries, PoolParty facilitates in silico experiments on genomic AI models in two ways. First, because sequence design is separated from sequence generation, new sequences can be generated on demand as the experiment proceeds. Second, the design card that PoolParty returns with each sequence provides metadata that can be used directly as covariates in downstream analyses, including surrogate modeling.

As a third example application of PoolParty, we carried out a surrogate modeling study of SpliceAI, a deep learning model of mRNA splicing. We asked how the insertion of a cryptic 5'ss near an existing (canonical) 5'ss affects the model's prediction for canonical site usage. Using the 5'ss of SMN2 exon 7 as the canonical site, we used PoolParty to construct a library in which cryptic 5'ss 9-mers of varying strength (i.e., MaxEntScan score [46]) were inserted at varying positions on either side of the canonical 5'ss (see Implementation). The specific values for cryptic site strength and position were recorded in the design cards (Fig. 4A).

Each variant sequence was paired with a matched control in which the cryptic site was disrupted, and the effect of each insertion was quantified as the difference in logit-transformed SpliceAI-predicted probabilities at the canonical site (Δlogit). Generalized additive models (GAMs) were used to separately model the effects of cryptic site insertion at exonic and intronic positions. Each of these two models has three terms, corresponding to the effects of position alone, strength alone, and interactions between these covariates (Fig. 4A).

The GAMs closely reproduced the observed data (Fig. 4B, C). Cryptic sites on the exonic side suppressed canonical site usage more strongly than those on the intronic side (Fig. 4D), with the effect peaking approximately 20 bp upstream of the canonical 5'ss. This is consistent with the observation that cryptic 5'ss are preferentially depleted near annotated ones on the exonic side [57]. Stronger cryptic sites drove greater suppression in both models (Fig. 4E). The effects of position and strength exhibited substantial

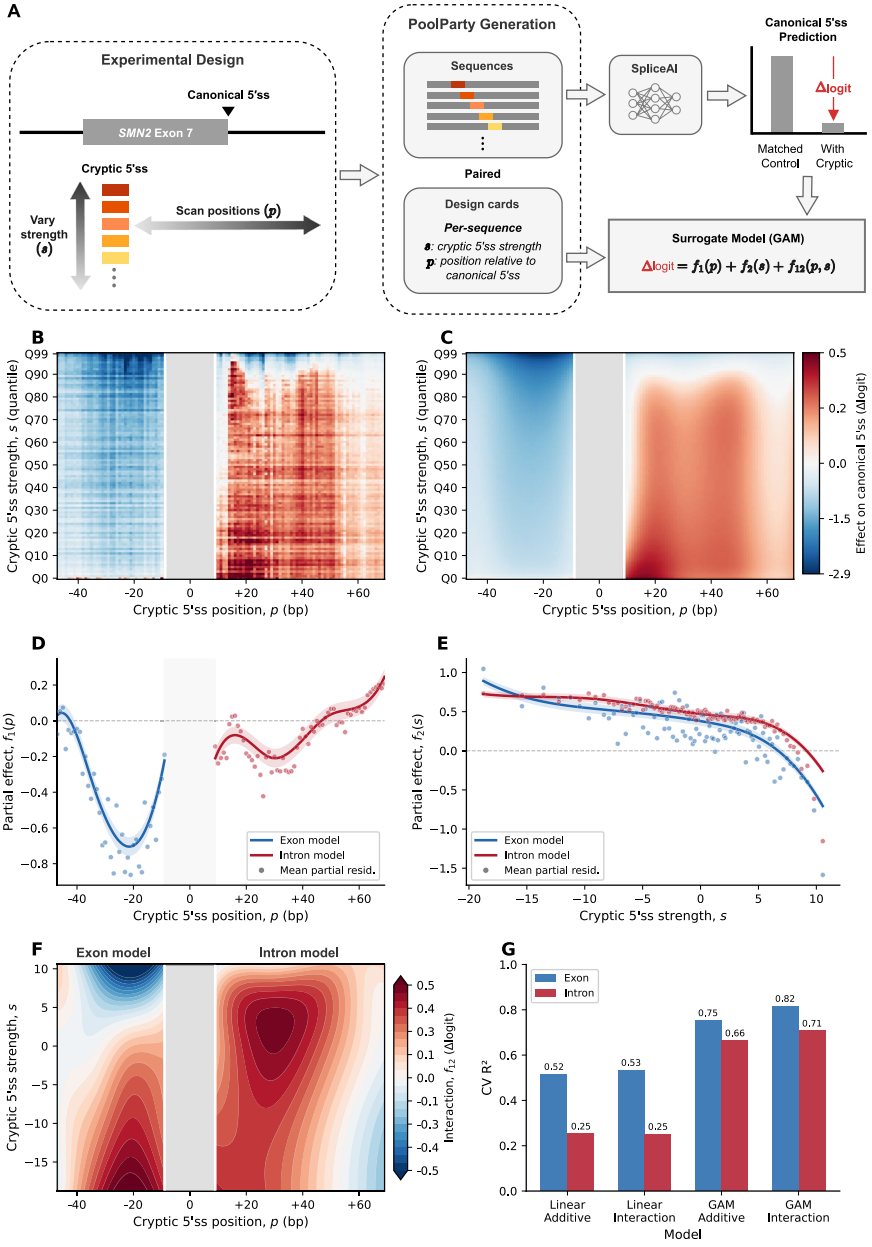


Fig. 4 Surrogate modeling of SpliceAI predictions. **A** Experimental design. Cryptic 5′ss 9-mers of varying MaxEntScan strength (*s*) were inserted at varying positions (*p*) flanking the canonical 5′ss of SMN2 exon 7. Each variant sequence was paired with a control sequence in which the splice-site GT dinucleotide is disrupted (GT → GA). PoolParty records the values of *s* and *p* in the design cards. The effect of each insertion on SpliceAI predictions at the canonical 5′ss is quantified as Δlogit and modeled as a function of *s* and *p* using a generalized additive model (GAM). **B** Mean Δlogit across the strength × position grid (100 × 100; each cell averages over 20 distinct 9-mers). The gray band marks the canonical 5′ss region. **C** GAM predictions (including the interaction term), using the same color scale as in B. **D** Partial dependence of the position term $f_1(p)$, shown separately for exon (blue) and intron (red) models. Shaded regions indicate 95% confidence intervals; points denote mean partial residuals. **E** Partial dependence of the strength term $f_2(s)$, with conventions as in D. **F** Interaction surface $f_{12}(p, s)$ for exon and intron models. **G** Checkerboard cross-validation R^2 for linear and GAM models, with and without interaction terms

interactions (Fig. 4F). Through a comparison to analogous linear models, we observed that the non-linear terms and interaction terms in the GAMs substantially improved predictive performance, with held-out R^2 values of 0.82 and 0.71 for exonic and intronic insertions, respectively (Fig. 4G).

Discussion

We have presented PoolParty, a Python package for designing oligonucleotide libraries. PoolParty facilitates this task by representing libraries as directed acyclic graphs (DAGs) that users can construct using a simple API. We demonstrated PoolParty through three use cases: a DMS library for protein GB1; an MPRA library probing transcriptional regulatory grammar; and a surrogate modeling analysis of the effects of cryptic 5'ss on SpliceAI predictions. These examples illustrate the flexibility of PoolParty and its applicability to both experimental and computational studies. Runtime and peak memory benchmarks for the three examples are provided in Supplemental Information (Fig. S2 and Table S1).

PoolParty was inspired by deep learning frameworks like PyTorch and TensorFlow, in which a high-level API is used to construct a DAG representing the desired neural network. The DAG then enables computation of model gradients via backpropagation. Similarly, in PoolParty, a backward pass through the DAG is used to decompose a state specific to each requested sequence into the specific Operations needed to construct that sequence during a subsequent forward pass through the DAG. This state-decomposition mechanism is a core innovation of PoolParty; in particular, it makes sequence generation random-access and allows users to test complex oligo pool designs without having to generate full sequence libraries.

We also highlighted the design card that PoolParty provides with each sequence it generates. Among other things, design cards facilitate using the parameters governing each sequence's construction as covariates in downstream analyses. Design cards are thus a bookkeeping device that allows users to avoid reverse-engineering sequence architecture from raw sequence when investigating how this architecture affects the output of a sequence-to-function model. For example, in the SpliceAI analysis of Fig. 4, the cryptic splice-site strength and position recorded in each design card provided the covariates used for surrogate modeling. We expect design cards to be increasingly useful as in silico experiments on genomic AI models become more common.

PoolParty complements existing library-design tools such as VaLiAnT [40], MPRAnator [41], and tangermeme [43] by providing a single framework for DMS libraries, MPRA libraries, and library designs that combine elements of both. Its DAG-based structuring of oligo pools lets users freely compose arbitrary sequence generation and transformation operations. Because the DAG encodes the full construction logic, structured design-card metadata are generated automatically as a byproduct. Additional comparisons between PoolParty and existing tools are provided in Supplemental Information, Figs. S3 and S4 and Tables S2 and S3.

PoolParty was designed to solve a specific computational problem: streamlining the design of DNA sequence libraries for a wide range of experimental applications and in silico studies. As such, it has several limitations. First, PoolParty operates on sequences, not genomic loci or transcript models. It can accept genomic coordinates as input to extract sequences from FASTA files, but does not interpret transcript annotations (e.g.,

in GFF/GTF files) or keep track of genomic coordinates through subsequent design steps. This may make PoolParty less helpful for certain genome-editing applications (e.g., as compared to VaLiAnT). PoolParty is also not a sequence optimization tool; it does not design sequences that have optimized codon usage [58], that are subject to complex synthesis constraints [59], or that have optimal activity as assessed by machine learning models [60]. PoolParty also does not address the problem of physically constructing DNA sequence libraries, e.g., by designing mutagenic primers [61]. We expect, however, that PoolParty can in many cases be productively paired with complementary tools that do address these challenges.

Conclusions

PoolParty streamlines the design of DMS experiments, MPRAs, and other multiplex assays of variant effect. As genomic AI models continue to advance, we expect that PoolParty will help researchers systematically probe and interpret the behavior of these models. Moreover, because PoolParty's composable design makes it straightforward to define new Operations, the framework can readily be extended to support novel assay types and analysis strategies as they emerge.

Abbreviations

5'ss	5' splice site
CRE	Cis-regulatory element
DAG	Directed acyclic graph
DMS	Deep mutational scanning
GAM	Generalized additive model
MAVE	Multiplex assay of variant effect
MPRA	Massively parallel reporter assay
PWM	Position weight matrix
TFBS	Transcription factor binding site

Supplementary information

The online version contains supplementary material available at https://doi.org/10.1186/s12859-026-06696-6.

Supplemental Information: Supplemental Methods, Figs. S1–S4, Tables S1–S3.

Acknowledgements

We thank Jack Desmarais, Evan Seitz, and Kai Loell for helpful discussions.

Author contributions

Z.L. conceived the project. Z.L. and J.B.K. designed the project, wrote the software, performed the analyses, and wrote the manuscript. Z.L. and A.C. wrote the documentation, developed the test suite, and deployed the software. J.B.K. supervised the research. All authors read and approved the final manuscript.

Funding

This work was supported by the National Institutes of Health [R01HG011787]. Computational equipment was supported by the National Institutes of Health [S10OD028632].

Data availability

No experimental data were generated or analyzed in this study. The sequence and software used in the paper were obtained from public sources referenced in the manuscript: the GB1 coding sequence from Olson et al. [17]; the MPRA backbone, TFBS sequences, and inert background sequence from Melnikov et al. [3] and Georgakopoulos-Soares et al. [56]; the SMN2 exon 7 genomic context from GRCh38.p13 (RefSeq assembly accession GCF_000001405.39); the human 5' splice-site position weight matrix from Wong et al. [45]; MaxEntScan from Yeo and Burge [46]; and the pretrained SpliceAI model ensemble from Jaganathan et al. [44].

Code availability

Project name: PoolParty. Project home page: https://github.com/jbkinney/poolparty-statetracker. Documentation: https://poolparty.readthedocs.io. Archived version: https://doi.org/10.5281/zenodo.19445098. Operating system(s): Platform independent. Programming language: Python (≥ 3.10). Other requirements: NumPy (≥ 1.20), pandas (≥ 1.3), beartype (≥ 0.22.9), pyfaidx (≥ 0.8.1), typing_extensions (≥ 4.0). The companion StateTracker package is installed automatically alongside PoolParty. License: MIT License. Any restrictions to use by non-academics: None.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 18 May 2026 / Accepted: 1 October 2026



References

1. Patwardhan RP, Lee C, Litvin O, Young DL, Pe'er D, Shendure J. High-resolution analysis of DNA regulatory elements by synthetic saturation mutagenesis. Nat Biotechnol. 2009;27(12):1173–5. https://doi.org/10.1038/nbt.1589
2. Kinney JB, Murugan A, Callan CG, Cox EC. Using deep sequencing to characterize the biophysical mechanism of a transcriptional regulatory sequence. Proc Natl Acad Sci USA. 2010;107(20):9158–63. https://doi.org/10.1073/pnas.1004290107
3. Melnikov A, Murugan A, Zhang X, Tesileanu T, Wang L, Rogov P, et al. Systematic dissection and optimization of inducible enhancers in human cells using a massively parallel reporter assay. Nat Biotechnol. 2012;30(3):271–7. https://doi.org/10.1038/nbt.2137
4. Kwasnieski JC, Mogno I, Myers CA, Corbo JC, Cohen BA. Complex effects of nucleotide variants in a mammalian cis-regulatory element. Proc Natl Acad Sci USA. 2012;109(47):19498–503. https://doi.org/10.1073/pnas.1210678109
5. Sharon E, Kalma Y, Sharp A, Raveh-Sadka T, Levo M, Zeevi D, et al. Inferring gene regulatory logic from high-throughput measurements of thousands of systematically designed promoters. Nat Biotechnol. 2012;30(6):521–30. https://doi.org/10.1038/nbt.2205
6. Mogno I, Kwasnieski JC, Cohen BA. Massively parallel synthetic promoter assays reveal the in vivo effects of binding site variants. Genome Res. 2013;23(11):1908–15. https://doi.org/10.1101/gr.157891.113
7. Levo M, Segal E. In pursuit of design principles of regulatory sequences. Nat Rev Genet. 2014;15(7):453–68. https://doi.org/10.1038/nrg3684
8. Inoue F, Ahituv N. Decoding enhancers using massively parallel reporter assays. Genomics. 2015;106(3):159–64. https://doi.org/10.1016/j.ygeno.2015.06.005
9. Vvedenskaya IO, Zhang Y, Goldman SR, Valenti A, Visone V, Taylor DM, et al. Massively systematic transcript end readout, "MASTER": transcription start site selection, transcriptional slippage, and transcript yields. Mol Cell. 2015;60(6):953–65. https://doi.org/10.1016/j.molcel.2015.10.029
10. Urtecho G, Tripp AD, Insigne KD, Kim H, Kosuri S. Systematic dissection of sequence elements controlling σ70 promoters using a genomically encoded multiplexed reporter assay in Escherichia coli. Biochemistry. 2019;58(11):1539–51. https://doi.org/10.1021/acs.biochem.7b01069
11. Klein JC, Agarwal V, Inoue F, Keith A, Martin B, Kircher M, et al. A systematic evaluation of the design and context dependencies of massively parallel reporter assays. Nat Methods. 2020;17(11):1083–91. https://doi.org/10.1038/s41592-020-0965-y
12. Romero IG, Lea AJ. Leveraging massively parallel reporter assays for evolutionary questions. Genome Biol. 2023;24(1):26 https://doi.org/10.1186/s13059-023-02856-6
13. La Fleur A, Shi Y, Seelig G. Decoding biology with massively parallel reporter assays and machine learning. Genes Dev. 2024;38(17-20):843–65. https://doi.org/10.1101/gad.351800.124
14. Fowler DM, Araya CL, Fleishman SJ, Kellogg EH, Stephany JJ, Baker D, et al. High-resolution mapping of protein sequence-function relationships. Nat Methods. 2010;7(9):741–6. https://doi.org/10.1038/nmeth.1492
15. Hietpas RT, Jensen JD, Bolon DNA. Experimental illumination of a fitness landscape. Proc Natl Acad Sci USA. 2011;108(19):7896–901. https://doi.org/10.1073/pnas.1016024108
16. Fowler DM, Fields S. Deep mutational scanning: a new style of protein science. Nat Methods. 2014;11(8):801–7. https://doi.org/10.1038/nmeth.3027
17. Olson CA, Wu NC, Sun R. A comprehensive biophysical description of pairwise epistasis throughout an entire protein domain. Curr Biol. 2014;24(22):2643–51. https://doi.org/10.1016/j.cub.2014.09.072
18. Adams RM, Mora T, Walczak AM, Kinney JB. Measuring the sequence-affinity landscape of antibodies with massively parallel titration curves. eLife. 2016;5:e23156 https://doi.org/10.7554/elife.23156
19. Findlay GM, Daza RM, Martin B, Zhang MD, Leith AP, Gasperini M, et al. Accurate classification of BRCA1 variants with saturation genome editing. Nature. 2018;562(7726):217–22. https://doi.org/10.1038/s41586-018-0461-z
20. Starr TN, Greaney AJ, Hilton SK, Ellis D, Crawford KHD, Dingens AS, et al. Deep mutational scanning of SARS-CoV-2 receptor binding domain reveals constraints on folding and ACE2 binding. Cell. 2020;182(5):1295–310.e20 https://doi.org/10.1016/j.cell.2020.08.012
21. Berger M, Philippakis A, Qureshi A, He F, Estep P, Bulyk M. Compact, universal DNA microarrays to comprehensively determine transcription-factor binding site specificities. Nat Biotechnol. 2006;24(11):1429–35. https://doi.org/10.1038/nbt1246
22. Maerkl SJ, Quake SR. A systems approach to measuring the binding energy landscapes of transcription factors. Science. 2007;315(5809):233–7. https://doi.org/10.1126/science.1131007
23. Noyes M, Christensen R, Wakabayashi A, Stormo G, Brodsky M, Wolfe S. Analysis of homeodomain specificities allows the family-wide prediction of preferred recognition sites. Cell. 2008;133(7):1277–89. https://doi.org/10.1016/j.cell.2008.05.023
24. Gordân R, Shen N, Dror I, Zhou T, Horton J, Rohs R, et al. Genomic regions flanking E-box binding sites influence DNA binding specificity of bHLH transcription factors through DNA shape. Cell Rep. 2013;3(4):1093–104. https://doi.org/10.1016/j.celrep.2013.03.014

25. Le DD, Shimko TC, Aditham AK, Keys AM, Longwell SA, Orenstein Y, et al. Comprehensive, high-resolution binding energy landscapes reveal context dependencies of transcription factor binding. Proc Natl Acad Sci USA. 2018;115(16):E3702–11. https://doi.org/10.1073/pnas.1715888115
26. Shen N, Zhao J, Schipper JL, Zhang Y, Bepler T, Leehr D, et al. Divergence in DNA specificity among paralogous transcription factors contributes to their differential in vivo binding. Cell Syst. 2018;6(4):470–83.e8 https://doi.org/10.1016/j.cels.2018.02.009
27. Khetan S, Carroll BS, Bulyk ML. Multiple overlapping binding sites determine transcription factor occupancy. Nature. 2025;646(8086):1001–11. https://doi.org/10.1038/s41586-025-09472-3
28. Kinney JB, McCandlish DM. Massively parallel assays and quantitative sequence-function relationships. Annu Rev Genomics Hum Genet. 2019;20(1):99–127. https://doi.org/10.1146/annurev-genom-083118-014845
29. Starita LM, Ahituv N, Dunham MJ, Kitzman JO, Roth FP, Seelig G, et al. Variant interpretation: functional assays to the rescue. Am J Hum Genet. 2017;101(3):315–25. https://doi.org/10.1016/j.ajhg.2017.07.014
30. Weile J, Roth FP. Multiplexed assays of variant effects contribute to a growing genotype-phenotype atlas. Hum Genet. 2018;137(9):665–78. https://doi.org/10.1007/s00439-018-1916-x
31. Avsec Ž, Weilert M, Shrikumar A, Krueger S, Alexandari A, Dalal K, et al. Base-resolution models of transcription-factor binding reveal soft motif syntax. Nat Genet. 2021;53(3):354–66. https://doi.org/10.1038/s41588-021-00782-6
32. Toneyan S, Koo PK. Interpreting Cis-regulatory interactions from large-scale deep neural networks. Nat Genet. 2024;56(11):2517–27. https://doi.org/10.1038/s41588-024-01923-3
33. Seitz EE, McCandlish DM, Kinney JB, Koo PK. Interpreting cis-regulatory mechanisms from genomic deep neural networks using surrogate models. Nat Mach Intell. 2024;6(6):701–13. https://doi.org/10.1038/s42256-024-00851-5
34. Seitz EE, McCandlish DM, Kinney JB, Koo PK. Uncovering the Mechanistic Landscape of Regulatory DNA with Deep Learning. bioRxiv. 2025;p. 2025.10.07.681052. https://doi.org/10.1101/2025.10.07.681052.
35. Nagai M, Murphy AE, Rizzo K, Koo PK. Toward generalizable and interpretable AI in regulatory genomics. Nat Genet. 2026 Aug;58(8):1787-1802. doi: 10.1038/s41588-026-02670-3. https://doi.org/10.1038/s41588-026-02670-3
36. Lee NK, Tang Z, Toneyan S, Koo PK. EvoAug: improving generalization and interpretability of genomic deep neural networks with evolution-inspired data augmentations. Genome Biol. 2023;24(1):105 https://doi.org/10.1186/s13059-023-02941-w
37. Cock PJA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics. 2009;25(11):1422–3. https://doi.org/10.1093/bioinformatics/btp163
38. Pereira F, Azevedo F, Carvalho Â, Ribeiro GF, Budde MW, Johansson B. Pydna: a simulation and documentation tool for DNA assembly strategies using python. BMC Bioinformatics. 2015;16(1):142 https://doi.org/10.1186/s12859-015-0544-x
39. Klie A, Laub D. SeqPro (Sequence processing toolkit); 2023. Software. Available from: https://github.com/ML4GLand/SeqPro.
40. Barbon L, Offord V, Radford EJ, Butler AP, Gerety SS, Adams DJ, et al. Variant library annotation tool (VaLiAnT): an oligonucleotide library design and annotation tool for saturation genome editing and other deep mutational scanning experiments. Bioinformatics. 2022;38(4):892–9. https://doi.org/10.1093/bioinformatics/btab776
41. Georgakopoulos-Soares I, Jain N, Gray JM, Hemberg M. MPRAnator: a web-based tool for the design of massively parallel reporter assay experiments. Bioinformatics. 2017;33(1):137–8. https://doi.org/10.1093/bioinformatics/btw584
42. Ghazi AR, Chen ES, Henke DM, Madan N, Edelstein LC, Shaw CA. Design tools for MPRA experiments. Bioinformatics. 2018;34(15):2682–3. https://doi.org/10.1093/bioinformatics/bty150
43. Schreiber J. tangermeme: A toolkit for understanding cis-regulatory logic using deep learning models. bioRxiv. 2025;p. 2025.08.08.669296. https://doi.org/10.1101/2025.08.08.669296.
44. Jaganathan K, Panagiotopoulou SK, McRae JF, Darbandi SF, Knowles D, Li YI, et al. Predicting Splicing from Primary Sequence with Deep Learning. Cell. 2019;176(3):535–48. https://doi.org/10.1016/j.cell.2018.12.015
45. Wong MS, Kinney JB, Krainer AR. Quantitative Activity Profile and Context Dependence of All Human 5′ Splice Sites. Mol Cell. 2018;71(6):1012–26.e3 https://doi.org/10.1016/j.molcel.2018.07.033
46. Yeo G, Burge CB. Maximum Entropy Modeling of Short Sequence Motifs with Applications to RNA Splicing Signals. J Comput Biol. 2004;11(2-3):377–94. https://doi.org/10.1089/1066527041410418
47. Servén D, Brummitt C. pyGAM: Generalized Additive Models in Python; 2018. Software (Zenodo). Available from: https://doi.org/10.5281/zenodo.1208723.
48. Otwinowski J. Biophysical Inference of Epistasis and the Effects of Mutations on Protein Stability and Function. Mol Biol Evol. 2018 10;35(10):2345–54. https://doi.org/10.1093/molbev/msy141
49. Otwinowski J, McCandlish DM, Plotkin JB. Inferring the shape of global epistasis. Proc Natl Acad Sci USA. 2018;115(32):E7550–8. https://doi.org/10.1073/pnas.1804015115
50. Tareen A, Kooshkbaghi M, Posfai A, Ireland WT, McCandlish DM, Kinney JB. MAVE-NN: learning genotype-phenotype maps from multiplex assays of variant effect. Genome Biol. 2022;23(1):98 https://doi.org/10.1186/s13059-022-02661-7
51. Faure AJ, Lehner B. MoCHI: neural networks to fit interpretable models and quantify energies, energetic couplings, epistasis, and allostery from deep mutational scanning data. Genome Biol. 2024;25(1):303 https://doi.org/10.1186/s13059-024-03444-y
52. King DM, Hong CKY, Shepherdson JL, Granas DM, Maricque BB, Cohen B. Synthetic and genomic regulatory elements reveal aspects of cis-regulatory grammar in Mouse Embryonic Stem Cells. eLife. 2020;9:e41279 https://doi.org/10.7554/eLife.41279
53. Smith RP, Taher L, Patwardhan RP, Kim MJ, Inoue F, Shendure J, et al. Massively parallel decoding of mammalian regulatory sequences supports a flexible organizational model. Nat Genet. 2013;45(9):1021–8. https://doi.org/10.1038/ng.2713
54. Grossman SR, Zhang X, Wang L, Engreitz J, Melnikov A, Rogov P, et al. Systematic dissection of genomic features determining transcription factor binding and enhancer function. Proc Natl Acad Sci USA. 2017;114(7):E1291–300. https://doi.org/10.1073/pnas.1621150114
55. Weingarten-Gabbay S, Nir R, Lubliner S, Sharon E, Kalma Y, Weinberger A, et al. Systematic interrogation of human promoters. Genome Res. 2019;29(2):171–83. https://doi.org/10.1101/gr.236075.118

56. Georgakopoulos-Soares I, Deng C, Agarwal V, Chan CSY, Zhao J, Inoue F, et al. Transcription Factor Binding Site Orientation and Order Are Major Drivers of Gene Regulatory Activity. Nat Commun. 2023;14(1):2333 https://doi.org/10.1038/s41467-023-37960-5
57. Dawes R, Joshi H, Cooper ST. Empirical prediction of variant-activated cryptic splice donors using population-based RNA-Seq data. Nat Commun. 2022;13(1):1655 https://doi.org/10.1038/s41467-022-29271-y
58. Swainston N, Currin A, Green L, Breitling R, Day PJ, Kell DB. CodonGenie: optimised ambiguous codon design tools. PeerJ Comput Sci. 2017;3:e120 https://doi.org/10.7717/peerj-cs.120
59. Zulkower V, Rosser S. DNA Chisel, a versatile sequence optimizer. Bioinformatics. 2020;36(16):4508–9. https://doi.org/10.1093/bioinformatics/btaa558
60. Schreiber J, Lorbeer FK, Heinzl M, Lu YY, Stark A, Noble WS. Programmatic design and editing of cis-regulatory elements. bioRxiv. 2025;p. 2025.04.22.650035. https://doi.org/10.1101/2025.04.22.650035.
61. Hiraga K, Mejzlik P, Marcisin M, Vostrosablin N, Gromek A, Arnold J, et al. Mutation maker, an open source Oligo design platform for protein engineering. ACS Synth Biol. 2021;10(2):357–70. https://doi.org/10.1021/acssynbio.0c00542

Publisher's note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.