AFRL-RI-RS-TR-2022-040

INTEGRATION OF TOP-DOWN AND BOTTOM-UP METHODOLOGIES FOR ACCURATE MODELING OF BIOLOGICAL NETWORKS

PRESIDENT AND FELLOWS OF HARVARD COLLEGE

MARCH 2022

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2022-040 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
CARL R. THOMAS
Work Unit Manager

/ S /
GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
MARCH 2022	FINAL TECHNICAL REPORT	SEPTEMBER 2017	SEPTEMBER 2021

4. TITLE AND SUBTITLE

INTEGRATION OF TOP-DOWN AND BOTTOM-UP METHODOLOGIES FOR ACCURATE MODELING OF BIOLOGICAL NETWORKS

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8750-17-C-0255	N/A	61101E
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
		R2DM

6. AUTHOR(S)

James J. Collins, George Steven, Jacqueline Valeri, Diogo M. Camach, Rani Powers

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

President and Fellows of Harvard College
1033 Massachusetts Ave, 5th Fl
Cambridge MA 02138-5369

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

DARPA
675 North Randolph St
Arlington VA 22203-2114

10. SPONSOR/MONITOR'S ACRONYM(S)

RI / DARPA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

AFRL-RI-RS-TR-2022-040

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

First, we developed “STORM/NuSpeak” as a tool to leverage big data and machine learning to automate the prediction and design of toehold switches more reliably. Second, we engineered “BioSeqML” as a biological sequence-based automated machine learning framework to automate development and interpretation of a range of machine learning architectures on biological data types. Finally, we created “DeepInducer” to generate novel inducible promoters in bacterial organisms, classify them, and predict their strength, all in a tunable manner. Furthermore, we’ve outlined some auxiliary work we’ve done in gene regulatory networks using various machine learning and statistical analysis methods.

15. SUBJECT TERMS

Machine Learning, Deep Learning, Computational Biology, Predictive Analytics, Synthetic Biology, Toehold Switches, Gene Regulatory Networks, Automated Machine Learning (AutoML), Natural Language Processing, Inducible Promoters

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	65

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
CARL R. THOMAS	N/A

PREVIOUS EDITION IS OBSOLETE.

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

Section **Page**

List of Figures

List of Tables

1.0 SUMMARY

First, we recognized that to screen a toehold switch requires a researcher to order hundreds of sequences to find a few with the desired output. We hypothesized that we could leverage big data and biological learnings to automate the prediction and design of toehold switches more reliably. To this extent, we've created STORM & NuSpeak for the optimization and remodeling of toehold sequences. STORM and NuSpeak enabled us to speed up the screening and fine-tuning of a single switch from a process taking potentially weeks, down to a few hours with a computationally assisted workflow. We've generated a comprehensive set of potential toeholds, which were then experimentally tested in the Collins and Church labs at the Wyss Institute.

Second, we sought out to enable researchers without considerable machine learning expertise to build statistical models. We've developed a framework called BioSeqML that automates machine learning for biological sequences. BioSeqML was created as a biological sequence-based automated machine learning framework to automate development and interpretation of a wide range of machine learning architectures on four data types. We've shown high performance on test sets typically requiring knowledge, time, and resources spent on data preprocessing, feature engineering, model architecture building, hyperparameter optimization, and model deployment.

Finally, throughout synthetic biology, there is an extant need for inducible promoters that are tunable and can respond to a variety of inducers. We created DeepInducer, a tool that can generate novel inducible promoters, classify them, and finally predict their strength. DeepInducer enabled us to create novel inducible promoters for desired organisms. With an input of the respective genome, and a desired inducer-transcription factor conformational change, DeepInducer can produce inducible promoters for that inducer-transcription factor combination, ranked by strength and probability.

2.0 INTRODUCTION

Advances in synthetic biology have changed paradigms in biotechnology engineering by deriving inspiration from nature. Researchers have isolated and adapted templates from naturally occurring circuit parts—such as inducible promoters, terminators, and riboswitches, but forward-engineering of components remains a challenge [1]. The workflow to develop a single biological circuit may require weeks of screening and fine-tuning to perform a desired function. As such, there is a strong need for in silico screening of circuit parts to ease integration of both naturally occurring and redesigned synthetic components into engineered biological systems.

2.1 STORM/NuSpeak

First, we will introduce STORM/NuSpeak—our work on the advancement of toehold switches as genetic circuit tools. To address the complexity of prediction and design of biological circuit parts, computational tools can aid in modeling and redesigning nucleic acid sensors, such as riboswitches. Since the discovery of naturally occurring riboregulators—RNA molecules that alter their translation rate in the presence of small molecules or nucleic acids via changes in secondary structure [2,3]—synthetic biologists have co-opted these circuit components for a variety of uses, from synthetic gene circuit construction to gene regulation [4-7].

The toehold switch, a particularly versatile synthetic riboregulator, can detect and respond to the presence of RNA molecules via linear–linear hybridization interactions [8]. The anatomy of a typical toehold switch consists of an unstructured RNA strand, followed by a hairpin that sequesters the Shine–Dalgarno sequence such that a downstream protein coding sequence is not translated when the switch is in the OFF state as seen in Figure 1(a). The toehold can be flexibly programmed such that the unstructured switch RNA and ascending stem of the hairpin (positions 1–30) are complementary to an arbitrary trigger 30-nucleotide RNA sequence—which upon hybridization to the switch subsequently melts the hairpin (toehold ON state), thereby exposing the Shine–Dalgarno sequence to the ribosome, which then initiates translation of the sequestered coding sequence. This fundamentally inducible nature of toehold switches has led to their successful use in both low-cost, freeze-dried, paper-based diagnostics as well as multiplexable components in complex genetic circuits with low crosstalk [9-12].

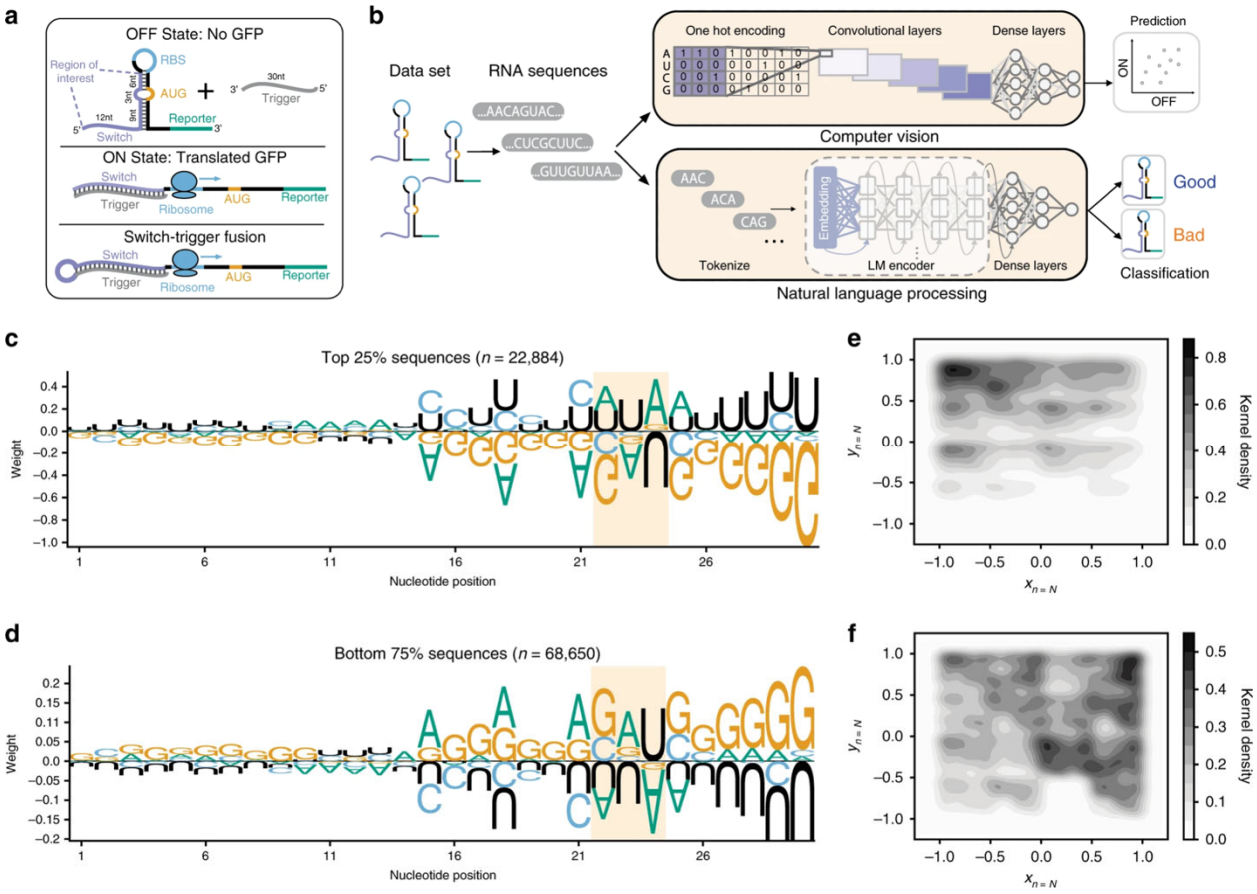


Figure 1. (a) Toehold switches modify their secondary structure in response to the presence of a complementary RNA molecule known as a trigger. (b) Two deep learning frameworks employing different strategies from computer vision and natural language processing were used to classify and predict toehold switch performance. (c) Sequence logos were calculated for the top 25% (N = 22,884) and (d) bottom 75% (N = 68,650) of sequences according to the experimental ON/OFF ratios. Highlight indicates the motif found in positions 22–24. (e) Chaos-game representations (CGR) for the top 25% (N = 22,884) and (f) bottom 75% (N = 68,650) of toehold switches were calculated to visualize macroscopic sequence patterns, where darker regions correspond to an enrichment of sequences in that CGR locus.

Sequence-based Toehold Optimization and Redesign Model (STORM) and Nucleic-Acid Speech (NuSpeak) are two orthogonal and synergistic deep learning architectures to characterize and optimize toeholds (Figure 1b). By applying techniques from computer vision and natural language processing, we 'un-box' our models using convolutional filters, attention maps, and in silico mutagenesis. Through transfer-learning, we redesign sub-optimal toehold sensors, even with sparse training data, experimentally validating their improved performance. This work provides sequence-to-function deep learning frameworks for toehold selection and design, augmenting our ability to construct potent biological circuit components and precision diagnostics.

2.2 BioSeqML

Next, we introduce BioSeqML— our work on the utility and deployment of automated machine learning tools, extended to biological sequence data. Automated machine learning (AutoML) refers to the iterative process of systematic design and deployment of machine learning (ML) pipelines with minimal user intervention [13,14]. End-to-end AutoML aims to provide domain expects with facilitated data pre-processing, feature extraction, architecture definition, hyperparameter optimization, model training, and performance evaluation [15], improving access to state-of-the-art machine learning solutions for a variety of fields. Even among experts, the adequate selection of algorithmic techniques and tuning of model parameters (i.e. typically ranging in thousands to hundreds of millions), constitutes a current bottleneck in machine learning design [16-18]. This barrier is especially relevant among researchers with limited programming or machine learning (ML) experience, where variety of design decisions can dramatically affect the quality and performance of the developed systems [15], Indeed, manual model definition and parameter optimization in ML requires considerable expertise and time to implement [13], while only providing limited value in the context of many applied fields (i.e. biology, chemistry, medicine), at least as compared to experimental data collection and prospective testing. For example, biotechnology researchers investigating the design space of functional Deoxyribonucleic acid (DNA), Ribonucleic acid (RNA) and Amino acid (AA) sequences using machine learning often struggle to produce predictive models under reproducible and scalable parameter selection criteria [1, 16].

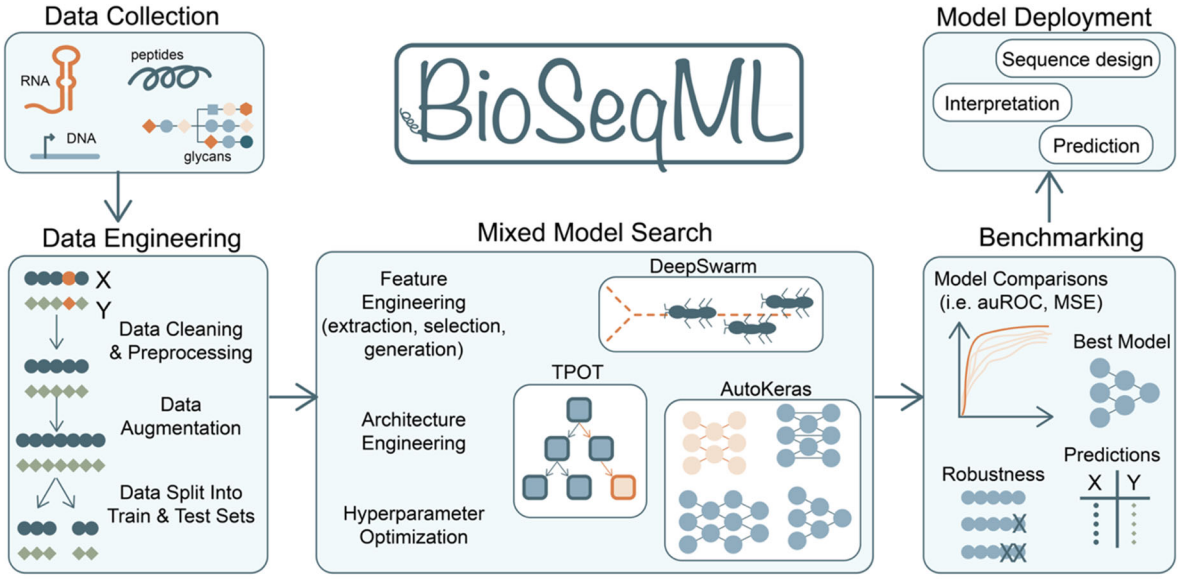


Figure 2. BioSeqML framework. Possible sequence data that end-users can utilize include toeholds, glycans, peptides, promoters, synthetic nucleic acids, and more.

While AutoML has become a desirable path to reduce the design burden of machine learning systems, current challenges in this area pertain to model benchmarking [19,20], with a need for systems that can integrate and evaluate a varied and comprehensive set of AutoML frameworks [21] while conducting automated algorithm and hyper-parameter search (AHPS). Recently, various promising Neural Architecture Search (NAS) solutions tailored for biological sequences

(i.e. DNA, RNA), such as BioNAS [22] and AutoGenome [23], have been reported with the goal of addressing a wide variety of predictive tasks for genomic sequence inputs with minimal user intervention. However, these solutions are limited in that they exclusively search among a reduced set of convolutional [22] or residual fully-connected [23] neural network (NN) architectures, in a framework that does not facilitate their integration with other search approaches. Indeed, many other non-NN AutoML tools exist that search through tree-models, such as XGBoost [24], with a rich variety of other NAS systems for neural network search, tree-based optimization with genetic programming [24], evolutionary [25] and even swarm intelligence for efficient architecture identification that are not currently used jointly in AutoML for biological research.

Thus, there seems to be a need for higher integration of multiple AutoML approaches within systems that handle data pre-processing, deployment and reporting for nucleic acid and biological sequence research. In response to this situation, we present a novel mixed end-to-end AutoML framework, optimized for building regression and classification models using nucleic acid sequence inputs. This system integrates multiple open-source AutoML tools with independent search mechanisms that allow for wider architecture search spaces than previously reported. We augment this integration with automatic data importing, pre-processing, architecture selection, hyperparameter search, model training, n-fold cross-validation, model deployment and performance reporting in an easy-to-use high-level application programming interface (API) as seen in Figure 2. Finally, we tested our integrated AutoML system through the automatic generation of machine learning models based on two canonical datasets for the prediction of RNA-guide activity in microbial clustered regularly interspaced short palindromic repeats (CRISPR) and CRISPR-associated (Cas) enzymes [26] with use in diagnostic applications [27].

2.3 DeepInducer

Finally, we introduce DeepInducer -- a tool that works to discover and optimize inducible promoters, currently within the space of bacterial organisms. Inducible promoters are promoters which react to some endogenous or exogenous stimulant, and in reaction, either begin expressing or repressing some downstream gene [28]. In our instance, and in the space of novel chassis design, we are primarily interested in the development of repressive inducible promoters -- that is, promoters that, when in the environment of a small molecule or other inducer, cease their repression and allow for a gene to begin its expression. This system effectively acts as an ON/OFF switch for gene regulation [29].

Inducible promoters at the prokaryotic level (excluding the added complexity of histamine, enhancer, and other influences in the case of mammalian cells [30]) consist of select cis-regulatory elements and trans-acting factors that describe much of the functionality [29]. As an example, one cis-regulatory element we explored was transcription factors (TF) -- proteins that controls the rate of transcription by binding to a specific DNA sequence, also known as a transcription factor binding site (TFBS) [29]. This component is especially relevant in the scope of designing inducible promoters as it is what allows a promoter to be controlled via ON/OFF signals. For our analysis, we used TF data from RegulonDB [31], verified through a corpus of literature.

Thus, knowing that inducible promoters can be constructed from interacting elements and factors, we propose a framework for the automated development of inducible promoters via the integration of the underlying components. We lay out this framework (in Figure 3) using deep learning technologies in the space of machine learning, as these tools are positioned well to learn, and generate novel samples, of non-linear relationships across sequence data. We propose a two-pronged approach, using natural language processing (NLP) methods that work to learn short and distant relationships in promoter sequence data and a generative adversarial network (GANs) that works to generate new inducible promoter samples with the same statistics as the training set. Generated/inferred inducible promoters also have to have in place some mechanism within the tool to classify them, in order to ensure we generated the right small molecule-inducer combination, as well as a mechanism to predict their given strength).

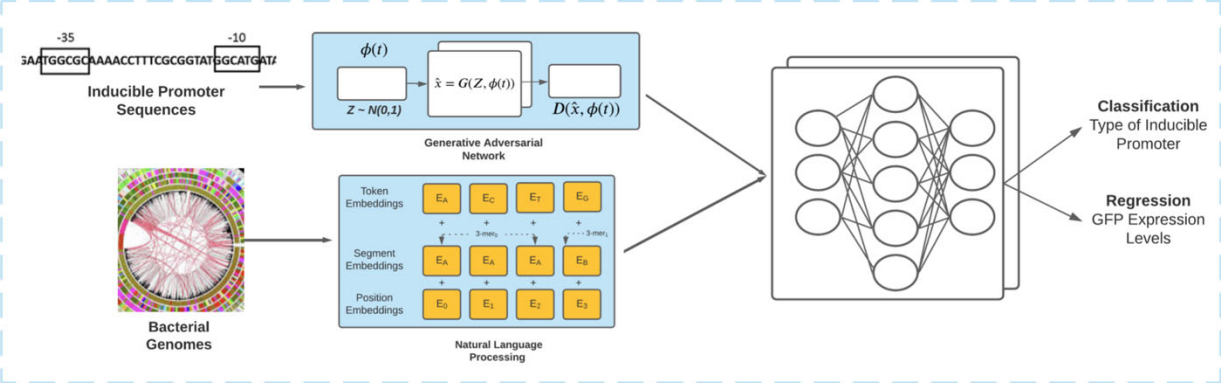


Figure 3. DeepInducer framework. Inducible promoters are used to directly train the generative adversarial network (GAN), while they are used at a later stage in the natural language processing (NLP) model during fine-tuning, after pre-training the model on an ensemble of bacterial genomes.

Even now, there is a substantial need to continue developing procedures in synthetic biology for designing biological circuit components. Through our design and discovery, we have been able to elucidate design rules for toehold switches (in STORM/NuSpeak), inducible promoters (in DeepInducer), and we have developed a sequence-based automated tool (BioSeqML) that deploys and functionally interprets a wide range of machine learning architectures.

2.4 Auxiliary Tools Developed

This section will be used a short outline on some of our early primer work within the SD2 program. Herein, we will delve into our *latent space representation models* and *nsmblR*, but only in the context of an introduction (covering both methods and results) so as to touch on these pieces of work, but keep explanations brief due to the limited scope and time investment of the respective projects. While these tools have had a shorter time span, they served as our original work into synthetic biology which led into some of the larger projects such as STORM/NuSpeak, BioSeqML, and DeepInducer.

2.4.1 Latent space representation models

Early in the SD2 program, we develop a variety of deep learning models inspired by neural embeddings as seen in Natural Language Processing (NLP). We developed a sequence embedding model that was used to embed amino acid sequences in the context of the ProtStab challenge. This model (*prot2vec*) converts amino acid sequences of variable length into a fixed-sized, low-dimensional, dense vector representation. This allows for us to apply any downstream statistical machine learning model on this space. We tested the protein embeddings on a large-scale protein family classification problem and obtained very high accuracies on the prot2vec space by doing simple k-nearest-neighbor classification, with results more predictive than those from Rosetta.

In the same vein, we build a model (*exoli2vec*) for learning the regulatory aspects of E. coli. Starting from information publicly available from resources such as RegulonDB [31] for curated interactions between transcriptional regulators and other genes, including causal events such as regulator binding to promoter regions upstream of target genes, we embed the more robust (highest confidence) instances of these network ontologies as well as the various sequences corresponding to the entities in these ontologies into ecoli2vec. The learning algorithm tries to keep entities that are closely related close in this vector space, while keeping entities with seemingly no relationship far in this space. As such, every biological entity as well as relationship is represented by a point (vector), with vector addition being the operator that composes points. This embedding allows one to perform simple nearest neighbor queries. As an example, we can infer novel relationships for a given transcriptional regulator X_i by examining the predictions Y_i of what would be the genes regulated by X_i. Employing this strategy, we have achieved high accuracies on predicting weak regulatory relationships present in RegulonDB and these queries of the vector space can lead to discovery of new relationships and thus genetic modules.

2.4.2 nsmblR

During the SD2 program we developed a novel approach to implement the identification of gene regulatory networks based only on transcriptional data using a “wisdom of crowds” approach [68]. This approach, which culminated in the development of the R package that we termed *nsmblR* and is available for the larger community both through TACC or at https://github.com/diogocamacho/nsmblR, incorporates a myriad of algorithms that use different mathematical and statistical techniques to infer biological networks. Of relevance, we have not only included algorithms that were previously developed in our lab [69] but also algorithms based on mutual information (ARACNe, MRNET), correlation networks (Spearman or Pearson, using the pcorr package), and regression (GENIE3). To achieve a true wisdom of crowds’ network, we employ a voting scheme that allows for the identification of a consensus model.

The implementation of all these algorithms span multiple platforms (R, MATLAB) and we have started the process of computationally utilizing these approaches. We have been focused on integrating the results of all the approaches into an R framework.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 STORM/NuSpeak

Toehold sequence generation—to define the sequences to be tested, we performed a sequence tiling on a variety of prokaryotic genomes, as well as the complete set of human transcriptional regulators. Briefly, each chosen sequence was tiled with a sequence length of 30 nucleotides and a stride of five nucleotides. Additionally, we generated a set of 10,000 random sequences of length 30, drawing each nucleotide from a uniform distribution at each position. This approach resulted in a set of 244,000 sequences to be synthesized and tested experimentally.

Data filtering and visualization—244,000 toehold sequences were tested by Angenent-Mari et al. [32] and the experimental data were obtained as logarithm-transformed GFP fluorescence measured at both the modified-ON state (with trigger present, fused to the switch sequence) and OFF state (without trigger). Measurements were normalized and quality control was performed as indicated in Angenent-Mari et al. [32], resulting in 91,534 sequences. All sequence logos were visualized with LogoMaker64. In addition, a final filtering step was applied prior to training the neural network, where sequences were split into 1000 bins for both ON and OFF distributions, and bins were down sampled to the mean number of counts across all bins (Supplementary Figure S1). The union of sequences from both the ON and OFF filtering stage was carried forward, resulting in 81,155 switches. Given remaining artifacts in the ON and OFF distributions, we would like to direct the reader towards experiments performed by Angenent-Mari et al.[32], where the authors evaluate the MLP model performance with a variety of methods, including re-sampling of under-represented continuous data points to achieve a balanced distribution, and find insignificant differences in models trained without this step. The CGR coordinates of the set were scaled between −1 and 1, and the third (length) dimension was omitted. These coordinates were subsequently plotted using a 2D kernel density estimate with the seaborn package to observe macroscopic sequence patterns.

Convolutional model architecture—The model was constructed of two convolutional layers to detect genomic motifs. The first convolutional layer consisted of 10 filters of width 5; the second convolutional layer consisted of five filters of width 3. The filters, or weight matrices, were convolved over nucleotide channels and pointwise multiplied with the input sequence, with the magnitude of this multiplication, or activation, corresponding to the degree of similarity between the filter pattern and the input [33]. Activations from the second convolutional layer were flattened into a one-dimensional vector and fed as input to three fully connected layers with successively decreasing numbers of nodes (150, 60, 15, respectively). All layers applied the rectified linear unit nonlinearity function to node outputs [34] and these activations were passed independently to two output layers: the ON and OFF prediction outputs, respectively. The last fully connected layer utilized linear activation to output continuous ON and OFF values. After each layer, a dropout rate of 0.3 was applied, and the ridge regression (L2 regularization) coefficient on the activations was set to 0.001 to decrease the risk of overfitting by constraining the values of the weights [34]. Errors between true and predicted ON and OFF values were computed over small batches of 128 toehold sequences per iteration using a mean squared error loss function. This loss information was back propagated through the model, and stochastic gradient descent was used to update the weights of the model such that the disparity between

model predictions and true value was minimized [34]. The Adam optimizer was used with a learning rate of 0.005 to train the model at a speed that achieved fast convergence without over- or under-shooting the optimal model fit. Weights were updated with respect to both ON prediction and OFF prediction simultaneously. Keras with a TensorFlow [35] backend was used to construct and optimize the model.

To assess the best architecture performance and train the final model, the data were shuffled and iteratively split using tenfold cross validation; the test set per fold was further split in half to be used as validation toehold sequences to select the optimal number of training epochs. A stratified split method enabled the cross-validation to be conducted with the class imbalance preserved in each fold. A deployable model was trained on 85% of the data, with 15% held-out as validation data to enable early stopping in training.

Hyperparameter optimization for convolutional models—Architecture design parameters were selected randomly rather than combinatorially as in a traditional grid search to enable a broader search of the architecture landscape within time and computation constraints. To find the optimal convolutional model architecture, several hyperparameters were varied and randomly sampled. The number of filters tested in the first layer were 5, 10, and 15, testing each combination with filter widths of 3, 5, or 7. The number of filters tested in the second layer were 5 and 10, with filter widths of 3, 5, or 7. We also tested a select number of architectures with filter widths of 10, 15, and 20, to ensure these shorter filter widths did not restrict the model accuracy. Additional hyperparameters, such as dropout rate, L2 regularization, and Adam optimizer learning rate, were varied to curtail overfitting and modify how the space of weights was explored during training: the dropout rates tested were 0.1, 0.3, and 0.5; L2 regularization values tested were 0 and 0.0001; and learning rate values tested were 0.0005 and 0.001. For all hyperparameter combinations, R^2 and Spearman correlation coefficients were evaluated across both ON and OFF values to ensure model predictions are sufficiently consistent with experimental results. Model simplicity was prioritized to avoid unnecessary complexity (i.e., more convolutional filters than needed) that did not augment the biological interpretability of the model. Five-fold cross validation was used to train and evaluate each parameter combination. For each fold, and for ON and OFF predictions separately, R2 and Spearman's rank correlation were calculated to estimate the generalizability of the model. The best architecture was selected by sorting the results by their combined average R2 over ON and OFF prediction, and choosing an architecture with first layer filters that would enable downstream interpretation of biologically-meaningful motifs and maximal ability to decode predictions in the context of toehold design rules.

Language model construction—The LM was trained on an in silico corpus of 4 million synthetic toehold sequences. We generated random 30 nucleotide sequences by sampling each nucleotide with uniform probability, and filled in the remainder of the 59 nucleotide sequence with the basic set of toehold structural rules (Figure 1a). These in silico sequences were sampled from a vast sequence space (430), and thus had little collinearity (Supplementary Figure S2), as evidenced by the fact that the distribution of pairwise edit distances are approximately centered at 22 nucleotides and 35 nucleotides for the switch region and complete toehold respectively. The basal NuSpeak architecture was derived from the PyTorch implementation of ULMFit [36] provided by fastai [https://docs.fast.ai/]. Custom tokenization and vocabulary functions were written using the fastai NLP wrapper in order to process genomic sequences as opposed to human languages. In brief, our modified ULMFit consists of a four-layer QRNN [37] with 1552

hidden activations in each layer, sandwiched between an embedding layer and a decoder layer. The LM classifier, likewise, consists of an embedding layer, a four-layer QRNN, and a linear classifier head, in lieu of the decoder. Dropouts for the output, hidden, input, and embedding layers were set to 0.15, 0.20, 0.30, and 0.05, respectively. Weight-decay dropout was set to 0.25 across all layers and weight-decay was set to 0.1 to provide strong L2 regularization during LM training. Most of these parameters were kept the same when training the LM classifier, with the exception of the output dropout, which was set to 0.5. Both the LM encoder and classifier were trained using the Ranger optimizer [https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer], which synergistically combines a Rectified Adam with the LookAhead optimization approach as described in Zhang et al. In addition, all models were trained using a batch size of 128 with back-propagation-through-time set to 60 in mixed-precision so as to maximize generalizability. A cosine learning rate schedule [38] was used during LM training for a total of 15 epochs, while a label smoothing cross-entropy loss [39] was used while training the classifier. Furthermore, to obtain a direction-agnostic view of toehold sequences, forwards and backwards models were separately trained and subsequently averaged during model prediction on the held-out test sets.

Language model evaluation—A random sampling of 5000 sequences from the experimental dataset were fed into the pre-trained LM encoder, which was trained to identify the top 25% vs the bottom 75% of toeholds. The 400-dimensional hidden state of the LM encoder was concatenated and averaged across all tokens for each input sequence in the randomly selected sample. For comparison, shuffled and scrambled input sequences, as well as random 59 nucleotide sequences, were also fed into the pre-trained toehold LM encoder. The shuffled input sequence was generated by shuffling the order of tokens in a tokenized input sequence and provides insight on the extent to which the model learns k-mer frequencies. Meanwhile, the scrambled input sequence was generated by first scrambling the real input sequence prior to tokenization, and offers insight on the extent to which the model learns nucleotide frequencies. We focused exclusively on classification tasks given that the architectures developed thus far in NLP are tailored for classification, though we note it would be possible to add a linear output layer for regression on top of the model discussed here.

In addition, to assess the impact of language model pre-training for the classification task, we built and trained a naïve classifier which also consists of an embedding layer, followed by a four-layer QRNN and a linear classifier head. Tokenized sequences from the experimental dataset were then directly fed into the naïve model without pre-training on in silico sequences. We observed that pre-training the LM on the in silico set of 4 million toeholds significantly stabilizes the model's word embeddings and yields statistically better performance on held-out test sets. To evaluate attention of the model, the fastai intrinsic attention function was used [https://github.com/fastai/fastai1/blob/master/fastai/text/interpret.py], which calculates the contribution of each k-mer in a sequence for the resulting gradient, given a classification prediction.

Language model comparison—We compared the language model to other NLP-based models. To compute tf-idf matrices, input sequences were tokenized in the same manner as for the language model, and tf-idf models were constructed using unigrams and bi-grams of three-mers (scikit-learn). The Naïve-Bayes, Logistic Regression, and Random Forest algorithms were used from off-the-shelf implementations provided in scikit-learn, along with a grid search used for

optimizing their respective hyperparameters. To compute skip-gram embeddings, input sequences were again tokenized into three-mers with stride 3, and a word2vec algorithm [40] [https://radimrehurek.com/gensim/models/word2vec.html] was used to learn contextual word embeddings across the entire experimental dataset. The pre-trained word embeddings were then used as inputs to an array of deep learning architectures as shown in Supplementary Figure S3, connected to a simple classifier head. Deep learning architectures were built using off-the-shelf implementations provided by Keras and Tensorflow. The self-attention layer was sourced from Keras [https://github.com/CyberZHG/keras-self-attention] for the attention-based architecture. All model specific hyperparameter details are enumerated in the source code files.

Saliency maps—Saliency maps were generated to visualize which positions and nucleotides mattered most towards high ON and low OFF model predictions [33]. The keras-vis package was used to analyze how small changes in a given input toehold sequence change the model's output predictions. The gradients were computed to highlight changes to the input sequence that produce large changes in the output predictions, revealing which positions in the toehold sequence were prioritized the most when predicting ON and OFF values. A saliency (i.e., an "importance score") for each position and each nucleotide at that position was calculated by summing the gradients across all positions and nucleotides for each toehold. This saliency was normalized by the number of times a given nucleotide appeared at that position to control for more frequent nucleotides.

Consensus model—For the consensus model, we first used transfer learning to freeze weights of both the NLP-based and CNN-based model, and re-trained both on a set of 168 toehold sequences from Green et al. for fine-tuning. For the CNN-based model, we modified the last output layer to output one value—the combined ON/OFF ratio alone—to be compatible with the normalized ON values reported in Green et al. An in silico external validation was conducted by comparing the ranks of the 24 Series A toeholds tested in Pardee et al. [10] with the ranks predicted by the original model, a model trained on Green et al. sequences [8] alone, and the transfer learning models. We then implemented a consensus algorithm to pick out toehold sequences that were both classified as good with high probability by the LM and predicted to yield high ON/OFF ratios by the CNN model. In brief, our simplex routine consisted of rank ordering by the geometric mean of the classification probability score and the predicted ON/OFF ratio score. We used the geometric mean over the arithmetic mean since it is more robust to extremes in either model. Of the sequences that passed through both models, eight "predicted good" and eight "predicted bad" were chosen to be experimentally validated. These eight test sequences were chosen at random amongst the list of sequences that passed both models.

Optimization—To optimize sequences using the consensus model (aka NuSpeak), we generated all possible variants by modifying the last nine nucleotides of each "predicted bad" 30 nucleotide sequence validated with the consensus model. The variants for each parent sequence were fed back into the pre-trained classifier to determine a classification probability score for each sequence, after which all sequences were fed into the pre-trained CNN regressor to predict the ON/OFF ratio. Our simplex routine was then used to identify variants that passed both sets of models and we selected eight optimized sequences for experimental validation. To optimize sequences using the STORM, we converted our CNN-based predictive pipeline to redesign poorly performing toehold switches via gradient ascent. The 20 toeholds with the lowest ON/OFF ratio were one-hot encoded and fed as inputs to the static model. A target ON/OFF

value of 1 was set and supplied to SeqProp, an open-source python package that enables streamlined development of gradient ascent pipelines for genomic and RNA biology applications [41]. Toehold design constraints were ignored during optimization, as the entire 59 nucleotide sequence was allowed to vary freely. After optimization, we "fixed" each sequence such that the modified toehold switch contained the conserved sequences and base pairing within the hairpin was preserved. We also offer the ability to incorporate these base pairing constraints in the loss function itself but found that the model sometimes settled into a local minimum and did not optimize effectively. At each iteration, the ON/OFF value of the initial toehold sequence was predicted and the difference between the predicted values and target values was computed. This discrepancy between predicted and target values was then propagated back through the model to update the input sequence in the direction that decreased the difference between the predicted ON/OFF value and the target. The updated toehold position weight matrix was used as input to the next round of optimization, and at the last round of iteration, the final sequence was composed of nucleotides with the highest probabilities in the position weight matrix. At each iteration, there is a balance between exploration and exploitation with an entropy parameter, as in some Bayesian optimization frameworks. Each sensor went through five rounds of optimization, and the resulting sequence with the highest ON/OFF value was chosen. Improvements in performance were quantified as the fold-change of the ON/OFF fold-changes between pre- and post-optimization.

Toehold switch validation—Toehold switch reactions were validated in cell-free protein synthesis systems as described in Pardee et al. [9] In brief, switches were individually ordered as 109nt Ultramer DNA oligonucleotides (IDT) consisting of a T7 promoter, 59 nucleotide switch region, common linker, and the first 5 nucleotides of a GFP sequence. Ultramers were added to a GFP gene (GFPmut3b-ASV with T7 terminator) through PCR amplification using Q5 high-fidelity polymerase (NEB). Resultant amplicons were treated with DpnI (NEB) to remove residual template DNA, and subsequently purified using a MinElute PCR purification Kit (Qiagen). Separately, triggers were synthesized from oligos consisting of antisense T7 promoter and antisense trigger sequence annealed to the sense strand of the T7 promoter using the HiScribe T7 Quick High Yield RNA Synthesis Kit (NEB). Reactions were incubated for 16 h at 37 °C, treated with DNaseI (NEB), and purified using the RNA Clean & Concentrator-25 kit (Zymo Research). Cell-free toehold switch reactions were performed using the PURExpress In Vitro Protein Synthesis Kit (NEB). Each reaction contained NEB Solution A (40% v/v), NEB Solution B (30% v/v), Murine RNase inhibitor (0.5% v/v; NEB), purified toehold switch PCR product (5 nM), and either no-trigger RNA (OFF state) or 10μM of trigger RNA (ON state). All reactions contained 5μL total volume and were carried out in triplicate within 384-well plates (Corning 3544) at 37 °C. GFP expression (485 nm excitation, 520 nm emission) was monitored on a plate reader (Molecular Devices SpectraMax M3) every 5 min for 150 min. OFF and ON values were taken as endpoint GFP fluorescence from reactions without trigger and with trigger, respectively. The GFP sequence used in this study can be accessed on Benchling [https://benchling.com/s/seq-Su3ovggGNrVRzsPxjAeA].

3.2 BioSeqML

Our mixed AutoML system was implemented in Python 3.6/7 with the capacity to handle both regression and classification tasks. This system operates by receiving a single comma-separated value (CSV) file with a list of nucleic acid sequences (with arbitrary constant length) and a target

value that the user wishes to predict for each of the indicated sequences. The data in the input file has to be specified such that the sequence column is labelled as “seq”, while the output column is indicated as “target”. After specifying such input file, the developed software interprets the input sequences and target values discarding other unspecified information. The input sequence alphabet is also automatically inferred and then used to generate one-hot encodings for all input sequences. Two independent pre-processing routines are also automatically applied on the target values depending on the prediction task defined by the user (i.e. regression or classification). For regression tasks, target numerical values are transformed to follow a uniform distribution between -1 and 1 using quantiles information to standardize model creation and evaluation. In the case of classification tasks for targets with float point numerical values, automatic or manual thresholding can be applied after quantiles transformation (with default threshold at 0.5). In classification mode, target classes are automatically inferred by the system through integer or text analysis of the target values to perform categorical binning with multiple classes allowed.

Following the pre-processing stage, an automated mixed AutoML model search is performed by sequentially implementing modified versions of AutoKeras [52], DeepSwarm [53] and Tree-based Pipeline Optimization Tool (TPOT) (54) to provide a joint AutoML framework for feature and architecture engineering. These three AutoML programs were selected due to their popularity of use, as well as the variety of architectures and search approaches covered by these systems. For instance, AutoKeras [52] is an open-source framework for neural network search based on model morphism and Bayesian optimization, where NN kernels and tree structured acquisition functions are used to iteratively search for optimal architectures. Conversely, DeepSwarm [53] performs NAS based on ant colony or swarm behavior algorithms, where a population of search agents using pheromone-like signals can sense local and global paths of previous explorations to collectively search for optimal NN architectures. Finally, the TPOT framework [54] built on top of Scikit-Learn, represents the third AutoML approach added to our mixed AutoML system to include tree-like architectures using genetic programming, feature engineering and self-learning algorithms [55]. The selection of AutoKeras, DeepSwarm and TPOT as frameworks for this initial proof-of-concept system, followed from the reported performance of these frameworks in recent benchmark evaluations across multiple datasets [27], as well as the capacity of these systems to operate in parallel mode on central processing units (CPUs) and graphic processing units (GPUs).

In order to integrate these independent AutoML systems into a single generalized framework, we generated a comprehensive set of functions that allow for all these systems to accept data from the same input and target structure, to generate analogous results that can be jointly evaluated and compared. Five additional use-defined parameters are also defined to guide overall system operation. These parameters include maximum run time per AutoML framework in minutes (i.e. max_runtime_minutes, default = 15), number of required cross validation folds (i.e. num_folds, default = 10), process text output (i.e. verbosity [0=None, 1=Minimum, 2=Extended, 3=Complete], default = 3), automatic class binning (i.e. do_auto_bin, default = True), and generate output backup folder (i.e. do_backup, default = True). All other internal search parameters specific to AutoKeras, DeepSwarm and TPOT, such as number of explored architectures, search methodology and early stopping routines are automatically selected based on the available search time selected by the user. Upon selection of the required input file and user parameters, our integrated mixed AutoML API conducts all subsequent data pre-processing, model search, training and evaluation to present analogous performance reports on the best

models found by each of the included AutoML frameworks. The operation of this integrated system was validated using two experimental datasets for RNA-guide functionality prediction in microbial clustered regularly interspaced short palindromic repeats (CRISPR) and CRISPR-associated (Cas) enzymes Cas13a and Cas13b used for efficient genome editing [56] and in-vitro diagnostics [27].

Mixed AutoML testing with LwaCas13a and CcaCas13b datasets —In recent years, a variety of software tools for the design of RNA-guides in CRISPR associated enzymes (i.e. Cas9, Cas12, Cas13) have become increasingly available to predict enzyme functionality based on ON-target and OFF-target activity [27]. Among the most valuable applications for these predictive analytics are those tailored to improve the diagnostic capabilities of the SHERLOCK (Specific High Sensitivity Enzymatic Reporter unlocking) technique [27], which combines target pre-amplification along with the CRISPR-Cas13 RNase [57,58] and CRISPR-Cas12 DNase for rapid and sensitive detection of nucleic acids. In particular, it has been reported that the tiling of SHERLOCK RNA-guides along various targets can led to significant variation in collateral activity, for Cas13a (an ortholog from Leptotrichia wadei bacterium - LwaCas13a) and Cas13a (an ortholog Capnocytophaga canimorsus Cc5 - CcaCas13b) [59].

The used LwaCas13a dataset contained 545 individual RNA guides of 28-nucleotides each, selected from six different biological origins (i.e. Dengue, Ebola, Zika). Conversely, the CcaCas13b dataset contained 729 individual guides of 30-nucleotides each from eight different biological origins. The guide activity values for LwaCas13a and CcaCas13b were obtained through detection tiling experiments where the zero-time point fluorescence was subtracted from the terminal fluorescence values. Such fluorescence values were used in raw format for AutoML training without further pre-processing, averaging, thresholding or binning for classification. During automatic thresholding guides leading to activity above the median were considered "good" guides (class=1), whereas the rest were considered "poor" guides (class=0). In classification mode, after automatic model selection and training all single-class receiver operating characteristic (ROC) and area under the curve (AUC) metrics during n-fold cross validation are reported along with the equally aggregated multi-class ROC average (i.e., macro) and the aggregated multi-class ROC average weighted by individual contributions of each class (i.e., micro). In regression mode, R^2 metric is reported along with logistic regression plots in n-fold cross validation. Final performance metrics for both classification and regression is reported by aggregating all predictions made during the n-fold cross validation process. After reporting, the deployment model is trained on all available data and exported to a working and backup repository.

3.3 DeepInducer

Finally, we will explore the methods, assumptions, and procedures surrounding DeepInducer, the tool we developed to generate, evaluate, and predict the strength and type of novel inducible promoters.

Inducible promoters, in the domain of bacterial organisms, consist of a few trans-acting factors and cis-regulatory elements that define the inducible promoter [60]. For example, in E. coli [61], we might expect observe the following influential elements: (1) transcription factors (TFs), usually discovered from RNA-seq/CHIP-seq data activating an operon (ex: araBAD, are what

endows the repressive quality of an inducible promoter—which allows researchers to harness and control the ON/OFF of genes via external signals); (2) sigma factors (an initiation factor that enables specific binding of RNA polymerase promoters)—analogous to eukaryotic factor TFIIB; (3) other design considerations such as TFBS/operator site location (distal vs. enhancer vs. a combination of both) and multiple operator sites [28], TF dimers engaging in homotetrameric protein interactions to form local DNA loops [28], and more. Building on these design principles, we were able to combinatorially enumerate different variations of these underlying elements and use machine learning models, such as natural language processing (NLP) [16] and generative adversarial networks (GANs) [49] to build up an inducible promoter from known, a priori information.

Figure 4. Sequence logos generated for one TF-inducer conformational relationship, in this case, TrpR-Tryptophan. These sequence logos are useful because they instruct us on which nucleotides generally belong in what position of the promoter, for that given promoter.

We first sought out to identify sequence logos for TFs of interest (which had an inducer conformational relationship) (Figure 4). These sequence logos and similarly, position weight matrices (PWM), will serve as the general zones to build around using the design architecture shown in Figure 5.

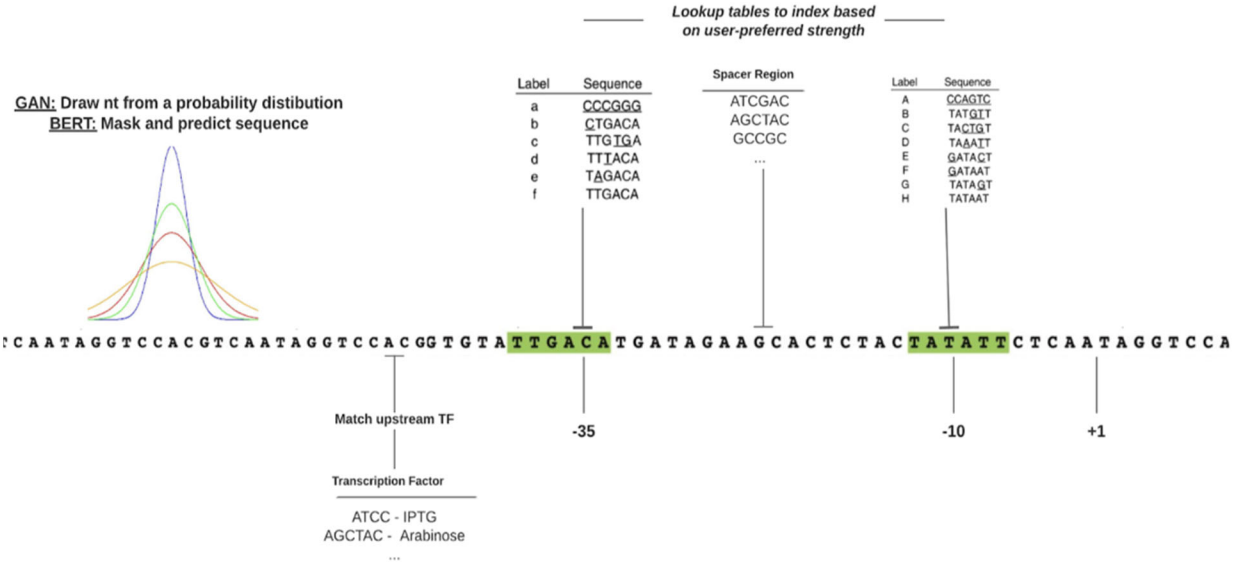


Figure 5. Combinatorial enumerations design and two different types of deep learning technologies, conditioned on the desired strength (determined by -10/-35 consensus sequences) and the desired level of repression (determined by the TFBS).

Transcription factors—proteins that control the rate of transcription by binding to a specific DNA sequence, also known as a transcription factor binding site (TFBS) [28]—are especially relevant in the scope of designing inducible promoters as it is what allows a promoter to be controlled via OFF/ON signals [28]. All TF data (N=2397) used in our analysis was obtained from RegulonDB [31]. Thus, these TFs can be used to create combinatorial designs alongside -10/-35 sequences, which typically dictate the overall strength of the promoter [29]. To this effect, we scrambled these components and ordered them in terms of overall similarity (using edit distance as our metric) to optimal consensus sequences. We had a few different sets to draw from: -10 regulatory sequences, -35 regulatory sequences, and transcription factors. For the -10/-35 sequences, we obtained the optimal sequences as known throughout E. coli promoter literature: {-10 optimal = TATAAT} and {−35 optimal = TTGACA} [29]. We then mutated these sequences and assigned rankings based on how far a mutated sequence edit distance (number of nucleotides required to obtain the original sequence either through substitution, deletion, or insertion) was from the optimal sequence. The further in edit distance, the larger the ranking – this means that ideally a promoter with both optimal -10/-35 consensus sequences would have the highest strength in GFP measure, and other combinations would only decrease strength. This methodology allows for filtering/look-up tables based on the strength of inducible promoters preferred. We then inserted the known TF for a given small molecule or otherwise inducer in the native location (a certain distance away from the TSS) to add repression/activation capabilities.

Transfer learning—Because most inducible promoters engineered today are E. coli based/derived, we wanted to extend the capability of using a vast range of promoters via learning from an (or multiple) organisms' genomes to inform our predictions for an inducible promoter's sequence respective to a given organism. Initially, we trained a BERT model to perform this step, but shifted to a simpler RNN model to test efficacy more quickly (such as the AWD-LSTM [63]). Training a model that takes in many bacterial genomes, effectively increases transfer learning capability [62] so that the 'language' of DNA can be learned and extended to new observations. We envisioned our bacterial genome ensemble to include E. coli genome assemblies, Pseudomonas aeruginosa, Pseudomonas fluorescens, and Bacillus subtilis to get a wide learning of bacteria-based organisms. We could then use these combinatorial designs, with fixed components, and predict in empty elements using our transfer learning model. Ideally, the transfer learning model would have base layers trained on the genomes, and a fine-tune head trained on specific inducible promoters in a certain organism regime, but this may not be feasible given the lack of breadth for inducible promoters in most organisms [55].

Training the machine learning models—We began by pre-training one of the primary neural networks that would be used to generate promoters using {one/few}-shot examples, the NLP component. We first decided to pre-train BERT [62] (due to its ability to handle unsupervised learning at-scale) on DNA sequences, consisting of slicing the E. coli genome randomly into sentences of 15-512 tokens (letters) [62]. This model was trained on a V100 Tesla GPU over ~7 days, for ~85000 steps. Additionally, we obtained proprietary SD2 (namely, NOT gate candidates) to use downstream fine-tuning. In tandem, we decided that the BERT model was too computationally expensive, especially considering we wanted to add a few other organisms into the mix. Thus, we shifted to using a AWD-LSTM which is a recurrent neural network that utilizes DropConnect for regularization, non-monotonically triggered averaged SGD (returning an average of the last iterations of weights), variational dropout, embedding dropout, weight

tying, and more [63]. This reduced our training time significantly and allowed us to effectively pre-train a language model using multiple gram-negative bacterial genomes. The AWD-LSTM, from FastAI, was chosen for its simplicity, reduced parameter and training cost, and robustness.

Strength prediction—We trained a deep vanilla neural network to regress on sequences of IPTG-induced promoters [28], consisting of TFs with various operator site configurations. This neural network was used to predict generated sequences' strengths, where end-users will be able to filter by high fold-change and high expression levels, all while minimizing leakiness [27]. We expect this component to be portable to users regardless of whether it is to be used specifically for inducible promoter strength prediction or to be extended to allow a user to train their own network. We provided the methods (Python3; PyTorch) that one-hot encoded a plethora of sequences, selected a target output, and overall trained the network in a supervised fashion [https://github.com/Wyss/SD2]. One critical component was to explore how accurately induction, non-induction, and the ratio between the two (fold-change) can be predicted to use it to 1) build inducible promoters with high RNA expression activity and 2) to have low induction/leakiness in order to drive induction in the ON state tightly. To this regard, we built a regression model consisting of 2 hidden linear layers, each with leaky-ReLU activations to avoid vanishing gradients. We measured induction, non-induction, and fold-change.

Generating sequences (NLP)—We utilized the output from the combinatorial designs and the transfer learning model to predict in the 'empty components' as seen in Figure 6, denoted 'XXXX'. Because this is a text generation problem, we instead utilized the decoder from the language model instead of adding subsequent layers onto the language model to perform binary or multi-class classification. We successfully accomplished this by taking a predicted token from the decoder output of the AWD-LSTM language model, appending a nucleotide, and building the sequence from there. We first start with the native TF, in its native location on the promoter sequence, and work upstream by reversing the sequence. We then re-reverse it and work towards the -35, then to the -10, then to the TSS, at each step continuously adding one token predicted from the entire sequence before it.

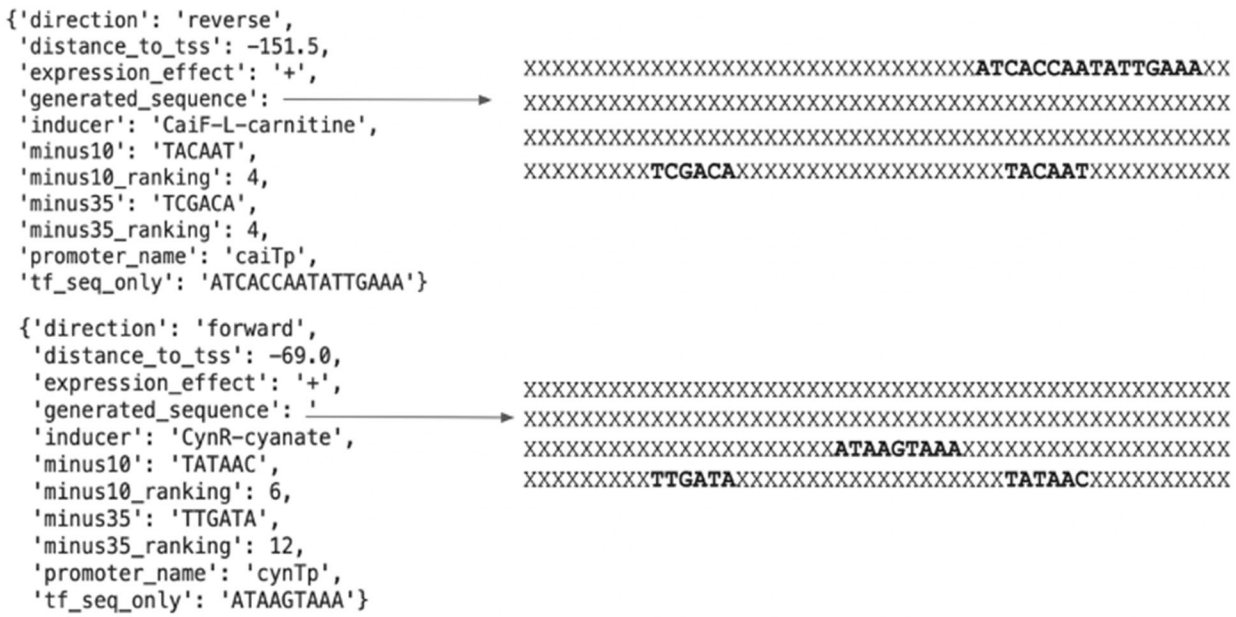


Figure 6. Two examples (CaiF-L-carnitine and CynR-cyanate) of combinatorial promoters containing major cis-regulatory elements but missing the rest of the promoter, which we later predict in using a language model.

Generating sequences (GAN)—For training a generative adversarial network to generate novel inducible promoter samples, we utilized an improved Wasserstein GAN (WGAN) [64]. The GAN generates IPTG candidates while the discriminative network evaluates the samples based on differences in probability distributions. The GAN learns how to map one-hot-encoded promoters from a latent space to the same latent (probabilistic space) where IPTG inducible promoters lie within. We chose to use a WGAN because penalizing the norm of the gradient of the critic with respect to its input allows for more stable training [64]. Because the GAN is generating samples based on the statistics of a given dataset, it is difficult to perform vastly different transfer learning operations as the statistics of other inducible promoters largely vary. One solution around this is to use conditional generative adversarial networks (CGANs) to instead condition the network on a label of a particular inducer, but this was not implemented in our current work.

4.0 RESULTS AND DISCUSSION

4.1 STORM/NuSpeak

Nucleotide over-representation in good and poor toeholds—A dataset of 244,000 toehold switches, including sequences tiled from viruses and the human genome as well as random sequences (see "Methods"), was designed jointly with Angenent-Mari et al. [32], with 91,534 switches meeting well-defined quality control criteria after experimental characterization. We conducted a traditional bioinformatics over-representation investigation and started by splitting sequences into the top 25th and bottom 75th percentile based on the experimental ON/OFF values. We then visualized each average position weight matrix as sequence logos, normalized with respect to the background nucleotide probability distribution (Figure 1c, d). Stratification into high- and low-performing sequences shows differential nucleotide composition immediately before the SD sequence, with uracil appearing over-represented and guanine under-represented in good toeholds (top 25%) and the converse true for poorly performing toeholds (bottom 75%). An enrichment for NUA in positions 22–24 is highlighted in the top-performing logo with 10.3% of the top 25% of sequences containing the NUA motif. This triplet corresponds to the three-nucleotide bulge directly opposite from the AUG start codon, suggesting that high-performing sequences may have an NUA at this bulge to prevent hybridization to the start codon (Supplementary Figure S1). Interestingly, 38.6% of top sequences contain a U in position 22 and 35.3% of top sequences contain an A in position 23, implying that single nucleotides belonging to the NUA motif are more prevalent than the complete motif. In addition, positions 15, 18, and 21 show over- and under-representation of adenine in bad and good toeholds respectively. These positions are directly opposite in the toehold stem to the first nucleotides of the only three in-frame coding triplets of the descending stem. This suggests that U in positions 51, 54, and 57 is detrimental to toehold performance, possibly since the N-terminus of the reporter protein cannot tolerate an in-frame stop codon (i.e., UAA, UGA), which would terminate translation of the reporter protein.

To elucidate any macroscopic sequence patterns between good and bad toeholds, we utilized chaos-game-representations (CGR) which provide an informative and lossless encoding scheme by which any nucleic-acid sequence can be fully represented with just three numbers (length, XCGR, and YCGR) (Figure 1e, f). Since all toeholds within our dataset have the same length of 59 nucleotides, a two-dimensional CGR vector, consisting only of XCGR, and YCGR, is sufficient to represent any given toehold sequence. Furthermore, since the trigger-binding region (first 30 nucleotides) is the principal variable region across all the toeholds, we computed its two-dimensional CGR coordinates, and observed an enrichment of A-rich codons amongst good performers (Figure 1e).

Biophysical properties are not predictive for top switches—The large size of the dataset allowed us to conduct an unbiased evaluation of toeholds' biophysical properties suggested by other studies [42]. As previous reports have indicated that GC content is important for the strength of the ON and OFF state stabilities [8], we compared the GC content distributions for top-performing sequences to that of all sequences (Supplementary Figure S4a). Our results suggest that successful toeholds may have a range of acceptable GC content between 20 and 60%, implying a necessity for some A–U base pairing in the switch.

Interpretability of deep learning framework predictions—As no single or combinations of biophysical properties were found to be sufficiently predictive of switch performance, deep learning sequence-to-function models using both CNNs and RNNs were built to predict toehold behavior. Given the recent advancements in both accuracy and accessibility of deep learning [34], a CNN [34] was constructed to take RNA sequences as inputs, employing two convolutional layers to identify plausible motifs and partial motifs in the input sequences (Figure 1b). Following the convolutional layers, the model employs a multi-layer perceptron (MLP) with three dense layers, where every node in each layer is connected to every node in the previous layer, to synthesize the features from the convolutions to output an ON and OFF prediction for each toehold sequence (see "Methods").

Given the vast number of weights and nonlinear functions that form the backbone of neural networks, it can be challenging to deduce why a model made the predictions it did [34]. While recent work such as soft explainable decision trees have enabled researchers to look inside this "black-box" by using a neural network to train a decision tree, we chose to visualize weights and activations of our trained model directly [33]. Taking inspiration from work in the fields of image recognition and genomics [34, 43], we "un-boxed" the first convolutional layer to visualize the features our model deemed important by interpreting the filter weights learned from input sequences as sequence logos (Figure 2a). To understand trends in convolutional filters, we trained the CNN 20 times and explored the frequencies of three-mers in the resulting ensemble of filters (Figure 2b). When compared to the expected value under a uniform distribution, the "CCC" three-mer occurs almost 2.5× more often than expected, suggesting the model may learn this motif for improved prediction. Additionally, the trained model learns to ignore sequences over-represented in the experimental toeholds—AGA and GAG, for example—that are part of the SD sequence and thereby conserved across all toeholds, indicating that the CNN learns to discern which positions are significant during the training process.

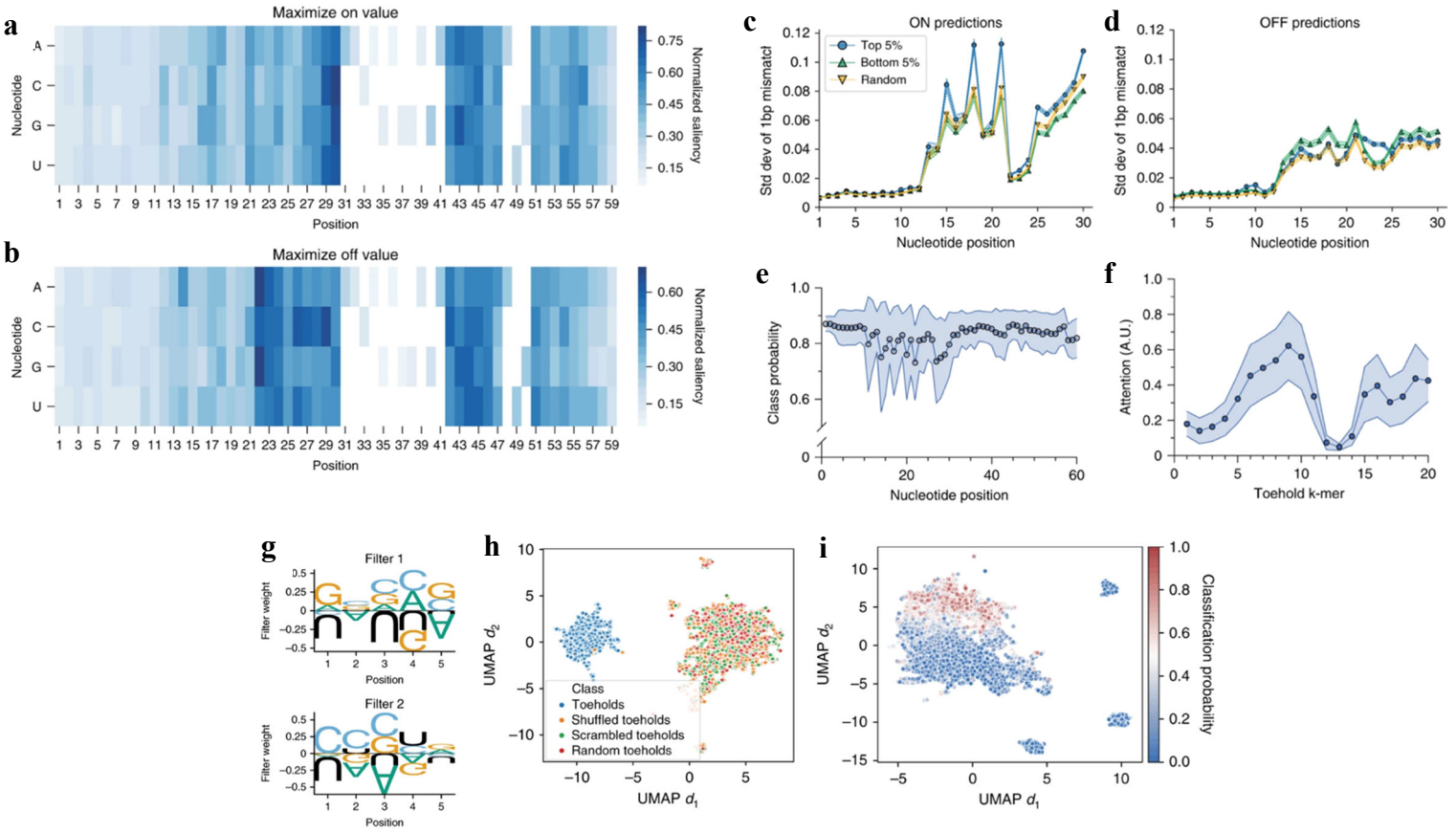


Figure 7. (a) In parallel, saliency maps were generated for 100 sequences to elucidate the importance of each nucleotide to the CNN-based model's ON and (b) OFF predictions, with saliency serving as a proxy for the model's attention to the nucleotide at that position. (c) To investigate the relative importance of each position in the toehold sequence, the effect of single base pair in silico mutations were evaluated on the CNN-based model for both ON and (d) OFF predictions, as well as for (e) language model predictions. For the CNN-based predictions, shaded area indicates the mean ± 95% confidence interval of 2500 sequences for the top 5% of sequences (blue circle), bottom 5% of sequences (green triangle), and a random sample of sequences (yellow triangle). For LM-based predictions, shaded area indicates the mean ± standard deviation of 5000 sequences. (f) White box tools were used to evaluate the "attention" of the language model for 5000 sequences randomly selected from the held-out test set, where attention represents the normalized contribution of each k-mer in a sequence for the class that the sequence belongs to. Shaded area indicates mean ± standard deviation. (g) To understand the sequence patterns registered by the convolutional neural network (CNN)-based model, the learned filter weights for each of the 10 filters of width 5 in the first convolutional layer can be visualized as sequence logos like position weight matrices, with two examples shown here. (h) The language model (LM) learns an embedding of synthetic toehold sequences (N = 5000) distinct from other nucleic acid sequences, including scrambled toeholds. (i) The LM embedding of 5000 experimentally tested toehold sequences cluster according to predicted classes.

Similarly, we also constructed an encoder-decoder architecture to learn the language of toehold sequences, where each k-mer is treated as a "word" or "token" and each toehold sequence is a "sentence". The LM encoder takes an RNA sequence that has been tokenized, or split into k-mers, as its input, and passes the tokens through an embedding layer, which maps each k-mer to a vector representation. The vector is then passed through a four-layer quasi-recurrent neural network (QRNN) [37] to return a 400-dimensional vector. In this LM pre-training step, the encoder learns a meaningful, context-dependent representation for each unique token found within the input corpus, which can then be extended with a linear classification layer to predict if a given toehold is good or bad. When augmented with a decoder that maps vectors back into tokens, the complete LM can generate meaningful sequences of arbitrary length within the language space.

In order to generate a sufficiently diverse and large toehold corpus, we first trained a LM on a set of 4 million synthetic toeholds generated in silico (see "Methods"). To determine whether the LM had learned a meaningful representation of toeholds, we mapped the 400-dimensional representation of a toehold sequence onto a reduced dimension manifold with UMAP [44] (Figure 7i) and compared to both scrambled and shuffled—which are scrambled after tokenization—input sequences, as well as random sequences. We observed no overlap between real toehold sequences and controls on the two-dimensional manifold, suggesting that the LM captured the importance of motif order in a toehold sequence.

Given the success of the LM classifier on synthetic sequences, we used the toehold dataset from Angenent-Mari et al. [32] to train a sequence classifier. The UMAP manifold of 5000 randomly sampled sequences color-coded by the predicted classes suggests that the classifier has learned to bifurcate good and bad performers (Figure 7h). Unsurprisingly, sequences with classification probabilities close to 0.5 populate the decision boundary. This classifier is ~3.7 fold and ~6.2 fold more predictive than classifiers that use shuffled and scrambled toeholds, respectively, again reinforcing the notion that sequence motif order is important for differentiating toehold performance and demonstrating that the model has learned more than k-mer frequencies.

Positional importance of nucleotides and model attention—To understand how variations in a toehold sequence might affect model predictions, we conducted a mutagenesis scan across sets of 2500 random experimental toeholds. For each position in the toehold, we mutated all four possible base pairs at each position and calculated the standard deviation of the CNN-based model's ON (Figure 7c) and OFF (Figure 7d) predictions. Spikes in effect sizes at positions 15, 18, and 21, mirror the important positions in the sequence logos (Figure 1c, d), suggesting the model learned positional importance of nucleotides. In parallel, a set of 500 randomly chosen good toeholds were serially mutated at each position with a random nucleotide and fed back into the LM to compute classification probabilities (Figure 7e). Echoing the prior mutational analyses, positions 26–30 were shown to have the biggest impact on toehold performance.

To ascertain the models' decision-making processes and further identify important regions in the toehold sequence, we first calculated the language model's intrinsic self-attention on a set of 5000 randomly sampled toeholds (Figure 7f). The self-attention map suggests that the ascending stem, specifically the last 12 nucleotides of the switch region, has the biggest influence on classification decisions. Unsurprisingly, given that the RBS and start codon do not vary across the dataset, the model learns to ignore these regions. These results are mirrored in the saliency

maps computed on the CNN-based model, where we evaluated the importance of each position in 100 random sequences towards maximizing the ON value (Figure 7a) and minimizing the OFF value (Figure 7b). Here, a higher saliency, computed by summing gradients across nucleotides at each position (see "Methods"), indicates that the nucleotide was considered to be more influential in the model's ON or OFF prediction process. To understand if the sequence saliency varies with the experimental values of the ON or OFF prediction, saliency maps for sets of high- and poorly performing toeholds were evaluated (Supplementary Figure S5). Poorly performing toehold maps show similarly low activation in the first 12 nucleotides as their high-performing counterparts, suggesting that the model learns the relationship between different regions of toeholds and predicted function.

Models predict toehold performance even with sparse data—Expanding the comparison between both model architectures on the same task, we systematically evaluated how the language model classifies good and bad toeholds for three ON/OFF thresholds (Figure 8a), in addition to how the CNN-based model predicts the ON and OFF states as continuous values (Figure 8b–d). Interestingly, we observed that all models had higher correlative metrics based on switch ON values alone. These results suggest that the models can learn features distinguishing a high ON value more readily, possibly resulting from variance in the OFF state due to autofluorescence not being subtracted out [8]. Since the ON/OFF ratio is an internally normalized performance metric and the fluorescence data were sorted into four bins, we chose an ON/OFF ratio of top 25% as the optimal threshold combination for the language model. As an additional validation experiment, toeholds corresponding to those obtained by tiling viral genomes were held-out during the classifier training phase. We subsequently fed these sequences into the trained model and scored the predictions and observed similar performance (average MCC~0.50) across toeholds sensing 20 different viral genomes.

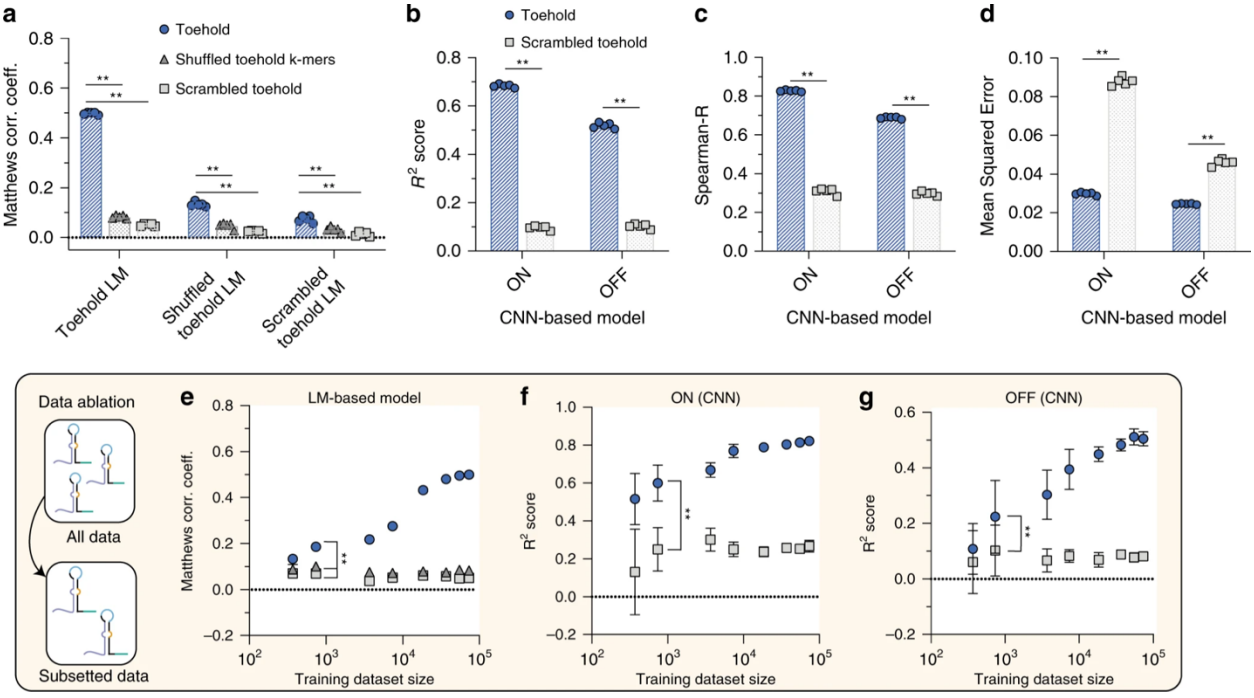


Figure 8. (a) The language model (LM) trained on toehold embeddings only accurately classifies real toeholds (blue circle), not shuffled (gray triangle, p = 2.45 × 10−14) or scrambled sequences (gray square, p = 5.03 × 10−15), as assessed by Matthews Correlation Coefficient (MCC). While language models trained on shuffled toeholds and scrambled toeholds are also more accurate for real than shuffled (p = 1.18 × 10−6, p = 4.41 × 10−4) and more accurate for real than scrambled toeholds (p = 1.58 × 10−8, p = 2.01 × 10−5), they fail to achieve the accuracy of the LM trained on real toeholds. (b) The convolutional neural network (CNN)-based model predictions for both ON and OFF are significantly higher than predictions on scrambled toeholds in five-fold cross validation, evaluated with R2 for ON and OFF (p = 2.97 × 10−14, p = 4.19 × 10−12), (c) Spearman correlation (p = 1.46 × 10−12, p = 9.09 × 10−13), and (d) MSE (p = 9.59 × 10−12, p = 2.12 × 10−9). For (a–d), N = 5 cross validation folds. (e) Data ablation studies were performed with both the LM and CNN-based model, evaluating both real toeholds (blue circle), shuffled sequences (gray triangle) and scrambled sequences (gray square). LM performance does not drop off steeply with half as much data and continues to perform better on real vs. shuffled (p = 9.23 × 10−9) and real vs. scrambled (p = 6.65 × 10−10) with just 736 training examples. For (e), N = 5 trials. f Similarly, the CNN-based model performance does not drop off steeply for either ON or (g) OFF prediction with half as much data and has significantly different R2 values at a training size of 736 samples for both ON and OFF predictions (p = 7.23 × 10−7, p = 0.028). For (f, g), N = 10 trials. For all panels, error bars represent mean ± standard deviation and all tests are two-tailed t-tests.

We also evaluated our models against more established, off-the-shelf methods (see "Methods"). When comparing the LM with other commonly used NLP architectures based on either term-frequency inverse document frequency (tf-idf) or skip-gram based word-embeddings, we observed that skip-gram-based word-embedding models were on-average ~1.8 fold more

predictive than the tf-idf models. The LM significantly outperformed all other word-embedding-based architectures, including a bidirectional LSTM paired with a self-attention layer, considered state-of-the-art for NLP sentiment classification tasks.

To elucidate whether the models were saturated, we computed learning curves for both sets of architectures (Figure 3e–g). Despite training on small dataset sizes (N = 736), both models were able to generate meaningful predictions (MCC~0.19 and R^2 ~0.6) relative to scrambled and shuffled controls. We hypothesized that the low variance associated with the language model predictions is a direct consequence of pre-training on the in silico set of 4 million toehold sequences, which results in a stable word-embedding, while the CNN is likely robust to smaller datasets due to its convolutional architecture [34]. Other commonly used word-embedding architectures had significantly higher variance in predictive performance across all training sample sizes and do not appear to saturate. Collectively, these data demonstrate the power of these architectures to train on considerably less data than anticipated.

Transfer learning and extending our models to unseen genomes—Considering the unique advantages of both the language model and CNN architectures, we incorporated both architectures into a pipeline for designing toeholds that optimally detect any arbitrary nucleic-acid sequence, such as a fragment of a viral genome (Figure 9a). While our models achieved robust performance on predicting toehold sensors tested by Angenent-Mari et al. [32], we sought to ensure their generalizability for scoring toehold sensors that detect free trigger RNA, given that the utility of toehold diagnostics applications stem from their ability to bind and detect exogenous RNAs. We thus explored our model performance on a smaller set of 168 sequences from Green et al. [8] that had been tested in a context containing free trigger RNA rather than with a fused trigger.

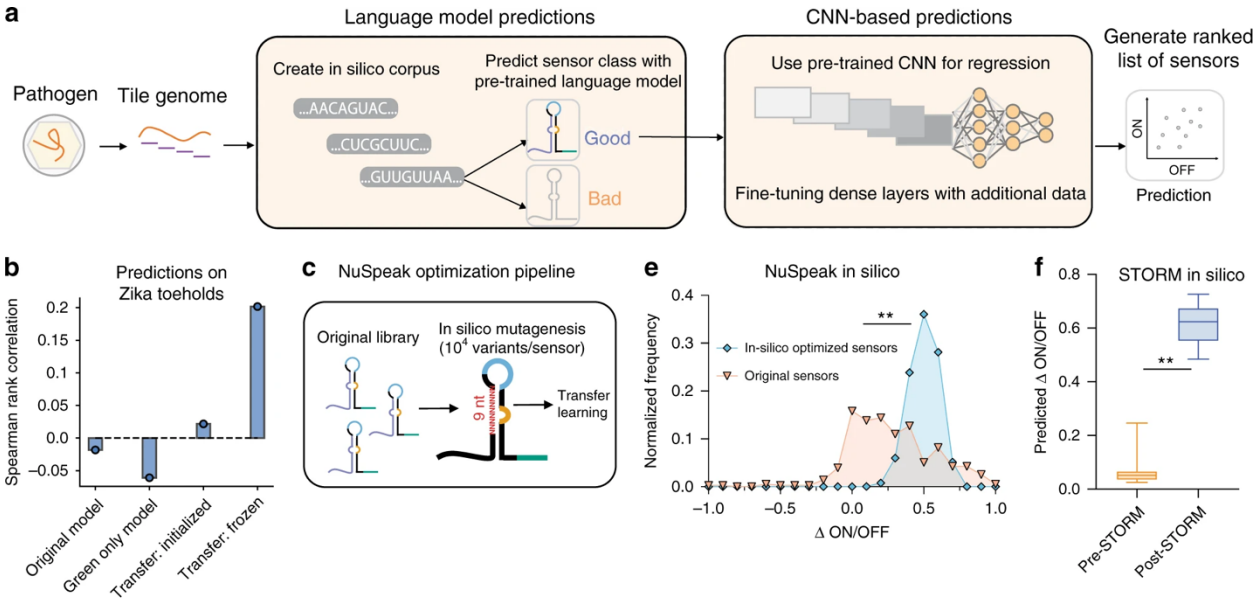


Figure 9. (a) Diagnostic sensor optimization pipeline involves tiling the genome of a pathogen, creation of an in silico corpus, and utilizing both the natural language processing (NLP)-based and convolutional neural network (CNN)-based models to generate a list of sensors for experimental validation. (b) To balance the utility of the large Angenent-Mari et al. [32] dataset with the nature of the Green et al. dataset [8], which tested toeholds with free RNA as opposed to switch-trigger fusions, we used transfer learning to fine-tune the existing model and achieve a higher degree of correlation with the predictions on an external validation set of free-trigger Zika toeholds (N = 24). (c) The NuSpeak optimization pipeline was designed to maintain base-pairing complementarity in the switch while producing all possible variants of positions 21–30. Sequences are then run through the consensus pipeline with fine-tuned models as above to produce a ranked list of sensors for a given region. (e) NuSpeak was used to optimize sequences with an increase in in silico ON/OFF ratio (blue circle) from the original ON/OFF ratio (orange triangle) for experimentally tested sequences (N = 354, p = 4.54 × 10−33). (f) Likewise, the Sequence-based Toehold Optimization and Redesign Model (STORM) was used to optimize experimental sequences, with center line indicating median and box limits indicating 25th and 75th quartile (N = 20, p = 3.33 × 10−8). For (e, f), a two-tailed Mann-Whitney U test was used. (g, h) Both pipelines can be used to predict and optimize SARS-CoV-2 viral RNA sensors, with experimentally validated toeholds performing as predicted. There is a significant increase in NuSpeak predicted bad (N = 8) and predicted good (N = 8) values (p = 0.026), as well as predicted good and optimized (N = 8) values (p = 0.020) and predicted bad and optimized values (p = 0.004). Likewise, there is a significant increase in STORM predicted bad (N = 4) and predicted good (N = 5) values (p = 0.010), as well as predicted good and optimized (N = 4) values (p = 0.089) and predicted bad and optimized values (p = 0.015).

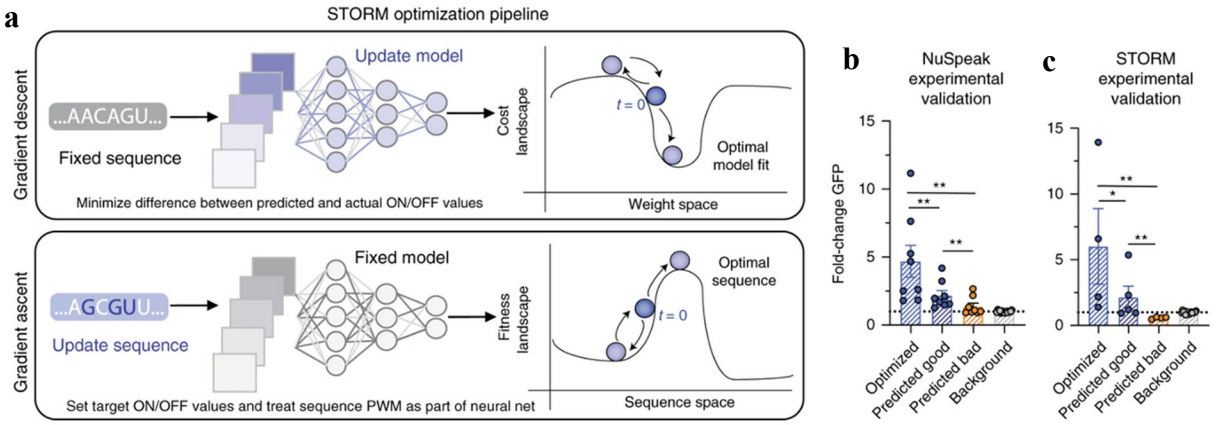


Figure 10. (a) In contrast to traditional model training, which uses gradient descent to optimize the model's weights from randomly initialized values (dark blue, time = 0) and predict the ON and OFF values for any given sequence, the CNN-based model can be inverted for gradient ascent of the sequence. Target ON and OFF values can be set, and the sequence space can be explored, starting at a given initialized sequence (dark blue, time = 0), to achieve those target values while the trained model remains fixed. Both optimization pipelines effectively improve the in silico ON/OFF ratio. For (b, c), error bars indicate mean ± S.E.M and a one-tailed Mann–Whitney U test was used. Background was calculated by measuring the initial ON/OFF ratio.

Because the pre-trained LM achieved a Matthew's Correlation Coefficient (MCC) of only 0.42 on the Green et al. dataset [8], compared with 0.51 on the Angenent-Mari et al. dataset [32], we hypothesized that a more predictive model could be constructed by fine-tuning our pretrained language model on toeholds tested in a free trigger context, using transfer learning techniques to bridge the gap. We incorporated the 168 free-trigger sequences from Green et al. [8] as our second, smaller training set. As research has shown that transferring weights from any number of layers can improve the accuracy of a re-trained model, we froze the first three QRNN layers of our pre-trained model, permitting only the weights in the fourth QRNN layer and classification layer to vary during training. This method achieved an MCC of 0.56 with a small held-out validation set, constituting a 33% improvement. As the CNN-based model showed a low R^2 of 1.87e-4 on ON/OFF values predicted for the Green et al. sequences [8], we then carried out the same transfer learning protocol for the CNN-based model by freezing the weights of the convolutional layers and allowing only the dense layer weights to vary during training. We observed an average R^2 of 0.18 and an average Spearman correlation of 0.36 over five-fold cross validation, again with small held-out validation sets.

We were also interested in the performance of the models on an external validation set. We evaluated the rank correlation of the CNN-based model predictions on 24 unseen Zika toeholds from Pardee et al. [10] (Figure 7b). When compared to our previous model, a model trained only on the 168 free-trigger sequences, and a transfer learning model initialized but not frozen with the weights of the pre-trained model, we confirmed that freezing the weights achieves the highest rank correlation of around 0.2 compared to the negligible correlations achieved by other models. Though these correlations may seem low, the experimental setup for the Pardee et al. sequences

[10] was vastly different: the researchers use lacZ activity as a readout and add an amplification step before detection so true trigger concentrations are unknown. Similarly, the language model accurately classifies all 24 toeholds as being good, consistent with the in-depth in silico screening process undertaken by Pardee et al. [10] to narrow a field of thousands of potential sequences to the 24 toeholds, based on rigorous secondary structure analyses, BLAST searches, and multiple constraints on the linear sequence. Unlike a model trained only on Green et al. sequences [8], which is likely underfit given the sparse training data, we can achieve meaningful biological predictions by conducting this fine-tuning step. Armed with these more predictive models, we integrated both the re-trained language model and re-trained CNN-based model into a pipeline that tiles any genomic sequence and returns a list of all possible toehold sensors ranked by their predicted ON/OFF value (Figure 9a).

To illustrate the value of our approach, and in a proof-of-concept demonstration to address the acute need for sensors that can rapidly detect emerging infectious diseases based on pathogenic genomic RNA, such as the current COVID-19 pandemic, we identified four regions of interest in the SARS-CoV-2 genome based on their uniqueness and orthogonality to other known human respiratory diseases [45]. We selected toehold sequences via consensus by both the LM and the CNN-based models (see "Methods"). We ligated each toehold sequence to a GFP reporter protein via a PCR reaction (see "Methods") to obtain a readily measurable readout upon toehold switch induction. We evaluated predictions experimentally by measuring the fold change in the fluorescence between the ON (trigger present) and the OFF (trigger absent) states. For both the consensus model and the transfer learning CNN alone (see above, Figure 10a,b), we found significant separation between "predicted good" and "predicted bad" sensors, consistent with our models.

Optimizing sequences with NuSpeak and STORM—Given limitations in current biological circuit design processes, we additionally built a framework to rationally redesign circuit components without maintaining complementarity to a trigger sequence. We converted our initial pre-trained CNN-based model to build a Sequence-based Toehold Optimization and Redesign Model, STORM (Figure 9d), by adapting the SeqProp method introduced in Bogard et al. [41] Rather than using gradient descent as in the previous classification and regression tasks, we used gradient ascent to optimize sequences to meet target ON and OFF values. To evaluate the utility of STORM, we optimized the 100 worst experimental toeholds (Figure 9f), with a significant increase in in silico predicted ON/OFF values after optimization. Saliency maps of pre- and post-optimization sequences (Supplementary Figure S6) reveal that the model attention decreases on the first 12 nucleotides for some optimized sequences, suggesting that optimization improves toehold performance by modifying select regions.

We sought to experimentally validate these results for both platforms by optimizing the sensors built from the SARS-CoV-2 genome. We measured an average fold-change of ~4.7 for sequences optimized with NuSpeak compared with an average fold-change of ~1.8 for the parent sequences, corresponding to ~160% improvement in sensor performance (Figure 10a). We also applied STORM to the four "predicted bad" SARS-CoV-2 viral RNA sensors and demonstrated experimentally that the optimized toeholds exhibited statistically significant increases in performance (Figure 10b), with improvements in performance of 28.4×, 1.45×, 9.66×, and 2.34×, respectively, for each of the four optimized toeholds. As the sequence optimization process may create a toehold that is not complementary with the original intended biological target, we

envision STORM being utilized as a valuable tool for unconstrained sequence development, such as in biological circuit component construction. As such, we built a website [https://storm-toehold.herokuapp.com] to make this prediction and redesign framework accessible to any interested researcher.

4.2 BioSeqML

Our first task was to collect example datasets (Table 1) for each sequence type (ranging from DNA, RNA, peptides, glycans, etc.) to test our models on. These datasets are sourced from a variety of papers and range in size from 9,982 to 91,534 sequences. We also include a synthetic dataset of nucleic acid sequences to ensure generalizability to all types of nucleic acid sequences and act as a positive control dataset.

We then tested performance of each of the three autoML frameworks on example datasets, for reference below (Figure 11). Each of the models is automatically compared to models trained on a scrambled control dataset to evaluate the effect of nucleotide frequency in predicting sequence target values. In Figure 11e, synthetic nucleic acids are scored by summing the nucleotides in the sequence, where A=1, T=2, C=3, and G=4. Any model should be able to fully explain and predict members of this example dataset. Additionally, we should not observe a decrease in performance on the scrambled control of the synthetic dataset.

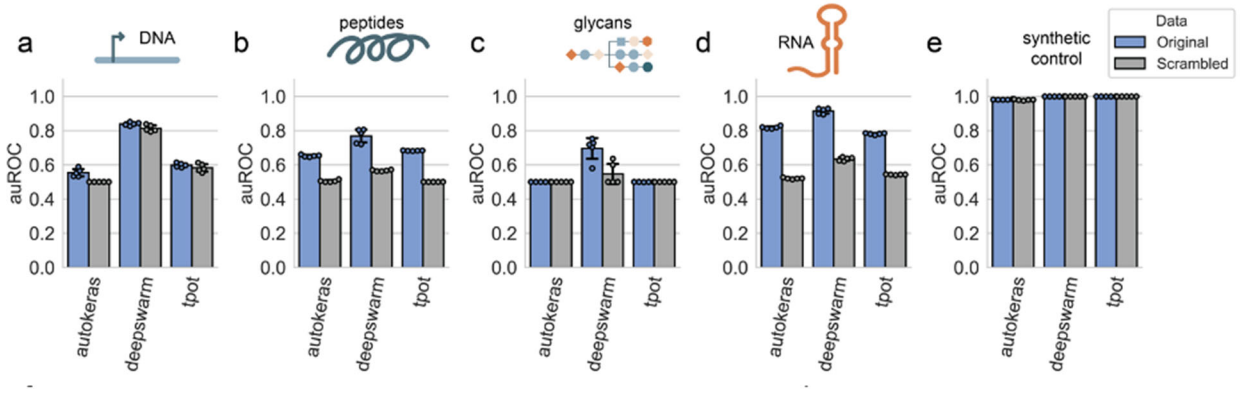


Figure 11. (a-d) AutoKeras, DeepSwarm, and TPOT tested on the original and scrambled data (randomized nucleotides) to evaluate the effect of nucleotide frequency in predicting sequence target values. (e) We tested our automated models on a synthetic nucleic acid sequence dataset to act as a positive control dataset and observe generalizability.

These models performed as expected. In most cases, we observe a drop in performance between the original data and the scrambled control, as tested in a five-fold cross validation setting. Models trained on the glycans dataset (Figure 11c) require further optimization, as DeepSwarm was the only model that performed above what we expect from a random model (auROC = 0.5). The glycans dataset poses unique challenges because the one-hot encoded vector has ~1000 dimensions due to the diversity of glycans. The synthetic nucleic acids dataset (Figure 11e) performed as expected, with nearly perfect auROC on both original and scrambled datasets.

Table 1. Datasets collected with sequence type, number of samples (N) and the target value to be predicted. Links to references are also included.

Dataset	Sequence Type	N	Target Value	Reference
Promoters	Nucleic Acids	9,982	Average promoter expression level	Deciphering eukaryotic gene-regulatory logic with 100 million random promoters
Peptides	Proteins	67,769	Enrichment of the sequence in phage display library	Antibody complementarity determining region design using high-capacity machine learning
Glycans	Glycans	19,992	Binary classification of glycan immunogenicity	Deep-Learning Resources for Studying Glycan-Mediated Host-Microbe Interactions
Toeholds	Nucleic Acids	91,534	ON/OFF value or ON value	Sequence-to-function deep learning frameworks for engineered riboregulators
Synthetic Nucleic Acids	Nucleic Acids	100,000	Summed score according to A=1, T=2, C=3, G=4	N/A

One automated component of the pipeline is the dataset robustness test (Figure 12). The user can specify that a dataset robustness test should be performed. If so, all three models are tested on decreasing fractions of the data and the resulting k-fold cross validation is plotted on a log scale. In this example, DeepSwarm appears most robust to decreasing dataset sizes. Models trained on scrambled controls appear mostly invariant to dataset size.

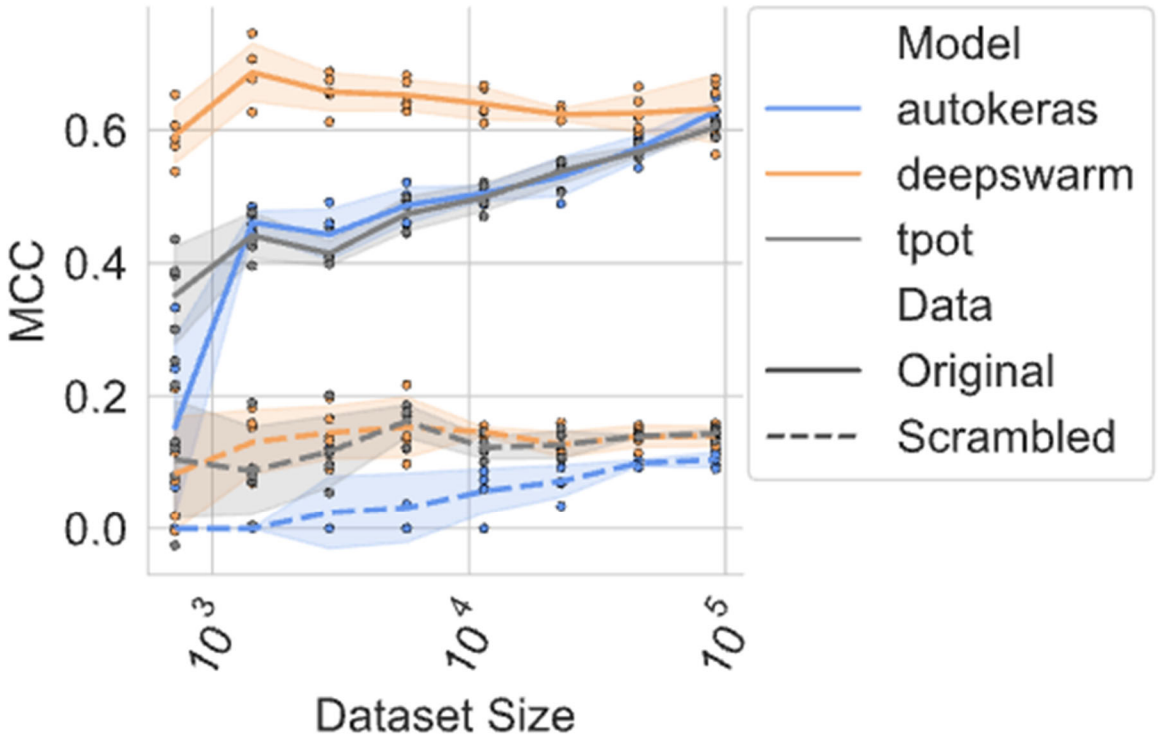


Figure 12. Model performance with decreasing dataset size. We observed that DeepSwarm generally will begin at a higher MCC but will not have performance increase with increasing dataset sizes, whereas TPOT and AutoKeras are both more sensitive to the dataset size and learn accordingly to its proportion.

We first sought to test DNA promoters (Figure 13a) and (Figure 13b) peptide sequences by running them through an in-silico mutagenesis protocol to identify positions in their sequences where changing that position had the largest effect on predictions. Error bars represent the mean of the prediction standard deviations plus/minus one S.E.M. for 500 samples of each class. We then sought to generate saliency maps to observe re-occurring nucleotides versus their respective positions (Figure 13c and Figure 13d). Since we knew that some positions did demonstrate strong re-occurrence, we computed corresponding sequence logos (Figure 13e and Figure 13f) from the DeepSwarm AutoKeras models for both promoters and peptides (100 of each) showing the effect of each alphabet member on the sequence. Similarly, we computed activation maps from the CNN output layers (Figure 13g and Figure 13h) and their corresponding sequence logos (Figure 13i and Figure 13j). These latter sequence logos were computed for the same models and data, showing cleaner sequence motifs and easier interpretation.

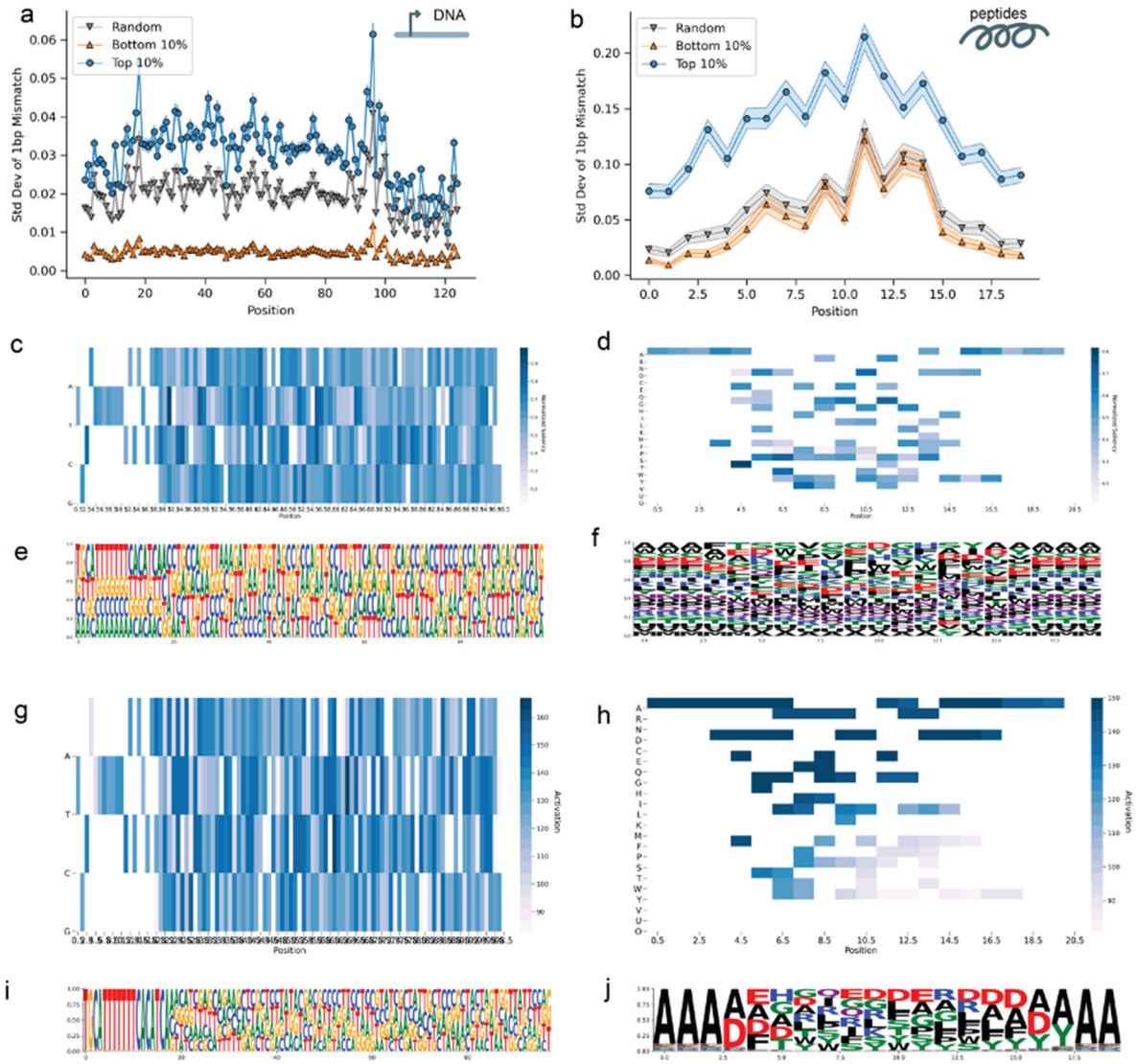


Figure 13. (a-j) Visualization techniques measuring standard deviation mismatching; sequence logos; and saliency maps for in-silico mutagenized promoters (a, c, e, g, i) and peptides (b, d, f, h, j).

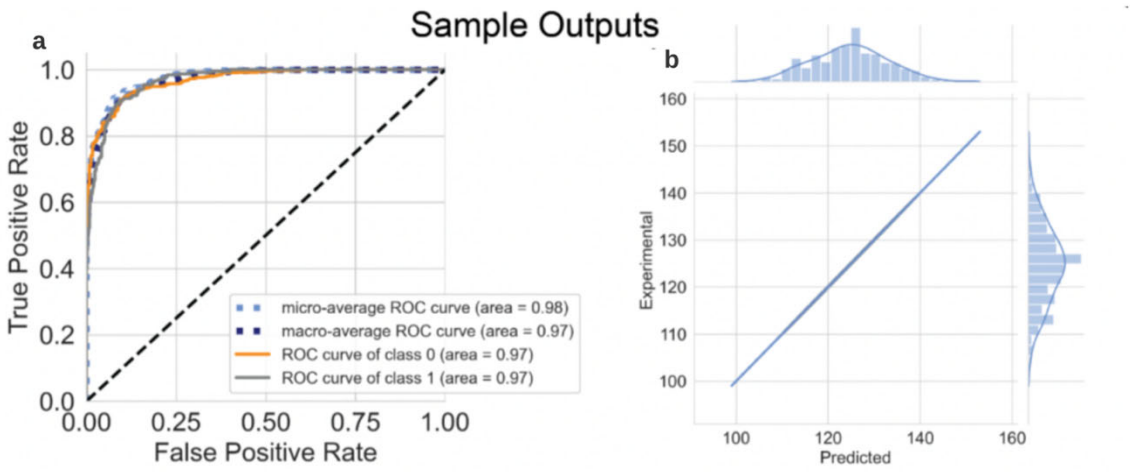


Figure 14. Sample outputs for BioSeqML after the computation pipeline is completed. (a) Classification task sample measuring ROC and micro-average ROC. (b) Regression task sample measuring the predicted versus experimental value. Both experiments were done on synthetic nucleotide controls.

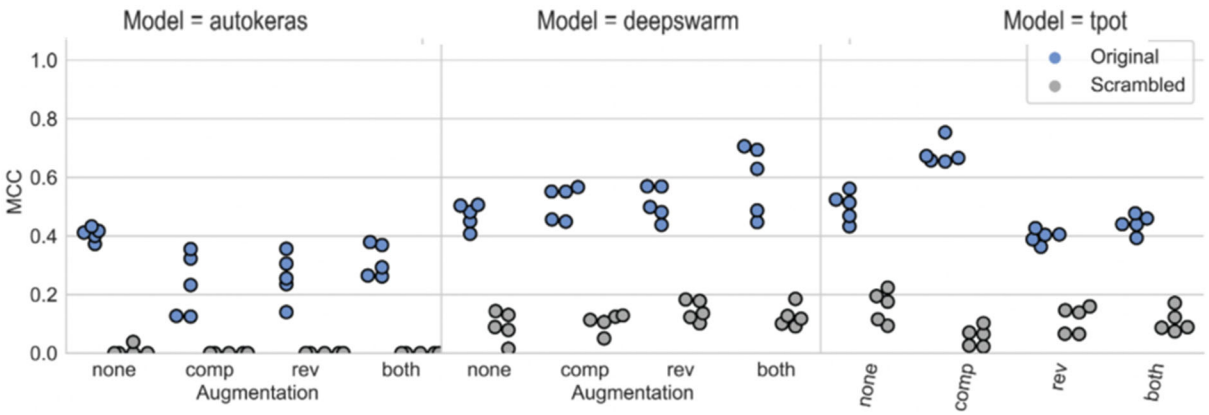


Figure 15. Augmentation methods for the various tools (AutoKeras, DeepSwarm, TPOT) of BioSeqML, measured by MCC (Matthews correlation coefficient) score.

We observed that by adding an augmentation option for nucleotide sequence-based datasets (here with a reduced 1K sample of riboswitches), we see that the complementation, reverse complementation, and both complementation techniques of augmentation achieve a higher MCC score for the DeepSwarm model than the original dataset (Figure 15). Thus, we may conclude that all three models are effectively distinguishing between scrambled sequences and the original sequences (Figure 15) and that certain augmentation methods, such as complementation, can increase classification performance (as measured by MCC), especially so in the case of TPOT.

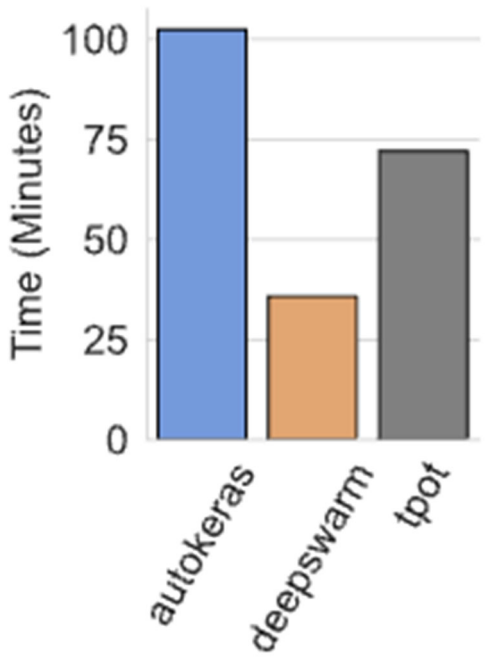


Figure 16. Time Needed to Evaluate Each Model (benchmarked on a 2.4GHz 8-core MacBook Pro).

Furthermore, it was important to evaluate the time needed to run these models end-to-end (Figure 16). Compared to the time for a researcher with machine learning expertise to build, train, and fine-tune models, the BioSeqML pipeline is a significantly more time-efficient option. The entire pipeline takes under four hours, at the end of which the user is left with a deployable model and many informative statistics. DeepSwarm consistently executes in the smallest amount of time, though we note that is likely due to the hyperparameter choice in the DeepSwarm architecture. A more in-depth hyperparameter optimization search will be tuned further in the next iteration of BioSeqML.

In the last years, state-of-the-art performances for various machine learning benchmarks have been achieved through manual tailoring of architectures and parameters that served the specific requirements of the intended prediction tasks [14,15,54]. However, automated architecture search and hyperparameter optimization has various advantages as compared to this manual model engineering; for instance, it can reduce the time and human resources required to apply machine learning in applied fields. Also, AutoML could improve the reproducibility and reduce untraceable bias in scientific studies using machine learning, with the added benefit of facilitating more objective comparisons among different datasets and analysis methods, which can only be fairly compared fairly when they receive the same level of human-implemented tuning [54,55].

4.3 DeepInducer

Because a critical component of DeepInducer is strength prediction (one of the last vanilla neural networks shown in Figure 3; the other network being a classifier), we built a regression model (as described in section 3.3) to predict GFP intensity and RNA expression (dependent upon the initial data training source, of which we used both for varying tasks [28,29]. We used IPTG induced promoters in E. coli, where N=14000 [28] for RNA expression and N=7000 [29] for GFP expression. We obtained an R^2 of 0.645 when comparing predicted vs. ground truth values in inducible expression, a R^2 of 0.577 for non-inducible expression, and a R^2 of 0.743 for the fold-change ratio (the ratio between inducible and non-inducible expression) as shown in Figure 17. These results inform us that we can accurately predict along these dimensions when filtering for high/low strength inducible promoters downstream, as well as filtering for promoters with low levels of leakiness (high GFP expression when induced, but low expression when non-induced, so that we can accurately control our promoter using logic).

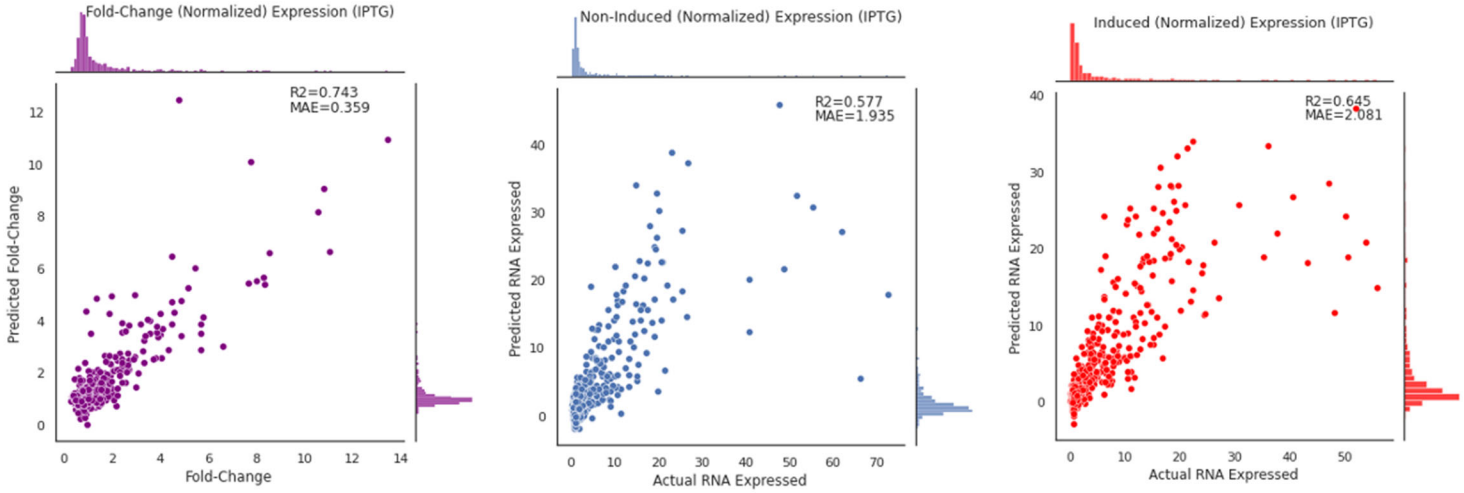


Figure 17. Fold-change, non-induced, and induced expression levels for various TFs (AraC, LacI, and more) and combinations in E. coli IPTG-induced promoters.

Further, we analyzed in depth the transcription factors available and derived thus far from literature review, as available through RegulonDB [31] for E. coli. We found that the transcription factors lie on some multi-dimensional spiral manifold when plotted with a dimensionality reduction algorithm (UMAP) [44]. We used the rogerstanimoto distance metric [65] and as a result we see a spiral manifold connecting the TFs (Figure 18). Further, we binned transcription factors by their length (<20, 20-30, >=30) and then observed their overall distance to the transcription start site (TSS). As shown in Supplementary Figure S7, both repressing and activating transcription factors form a bell curve centered a few base pairs to the left of the TSS. What is interesting is that with transcription factors longer than 30bp, we observe a much tighter proximity to the TSS, whereas the other bins appear to have a wider distribution around the TSS.

Strength prediction and promoter evaluation— Utilizing the GAN and NLP models, we generated promoters by fixing positions for certain nucleotides we knew as ground truth that type of promoter and then either allowing the generator to modify the sample in the latent space (in the GAN version) or sending the empty promoter into the NLP model and predicting the next token after our 'ground-truth' nucleotides. We then were able to use these results to observe the kernel density estimation (KDE) of predicted promoters' strength, and visually understand which promoters are more likely to express higher strength overall vs. promoters that may not be as highly expressing (see Figure 20). In Figure 20, we see that, for example, CynR-cyanate is more likely to be non/slightly expressive in an induced state and with a plateauing number of promoters that express low RNA expression.

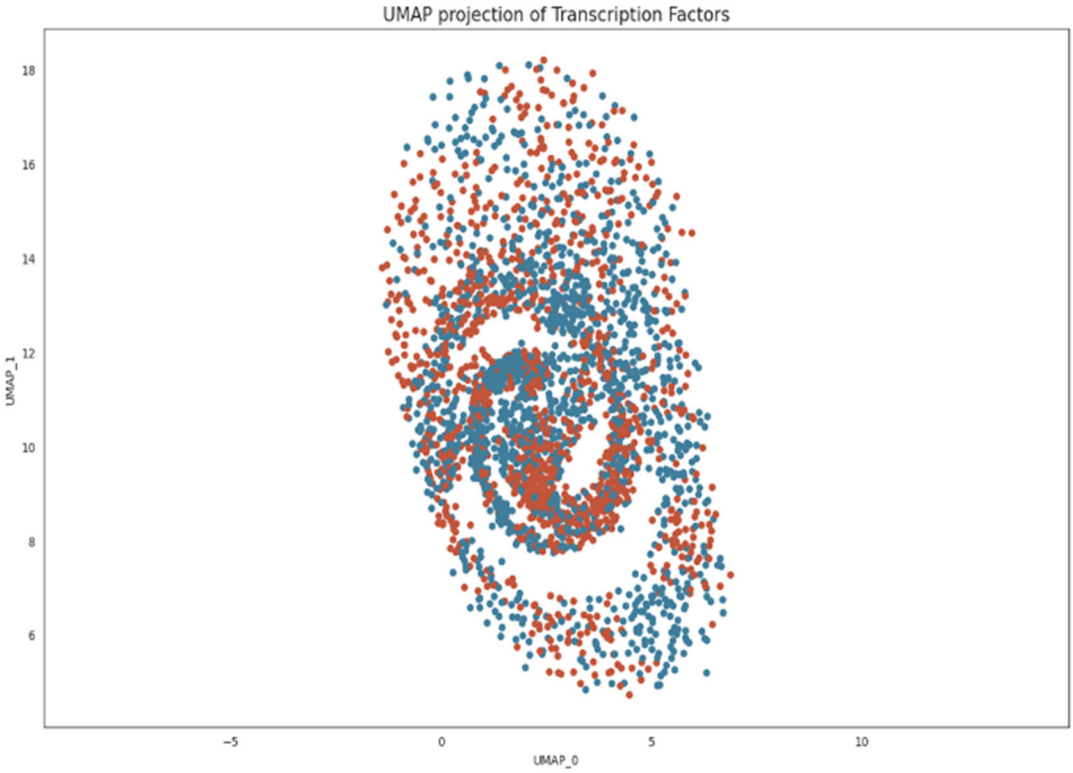


Figure 18. Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) performed on transcription factors in E. coli (N=2397), colored by their respective expression effect (repressive or activative).

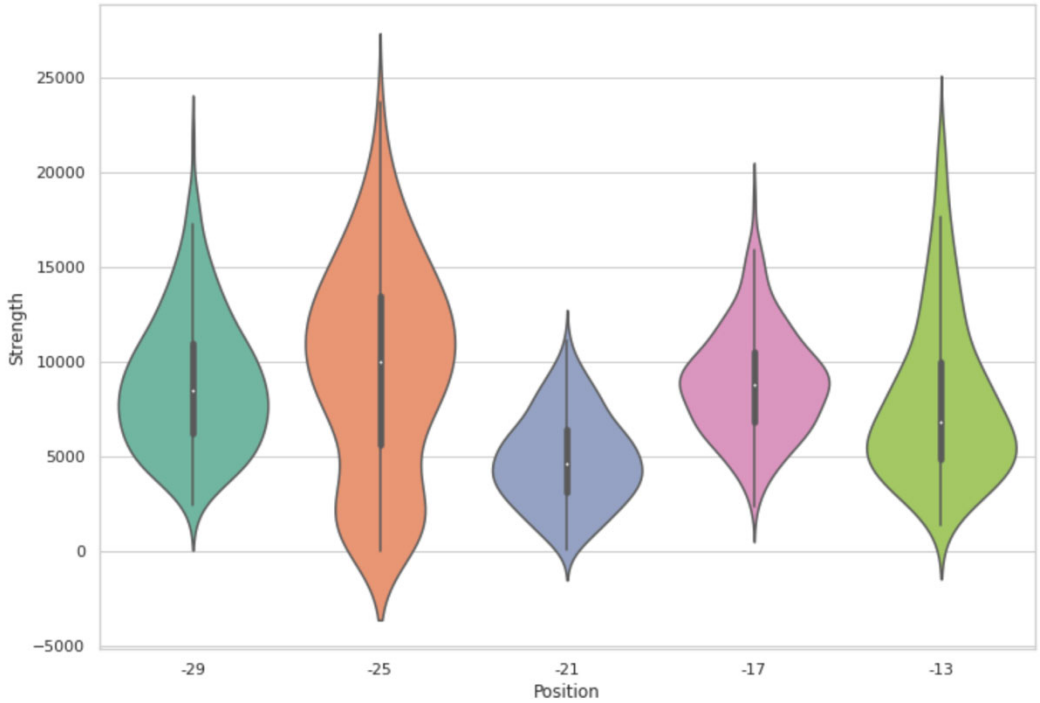


Figure 19. Predicted GFP expression for GAN-generated 3-mer samples of TetR in E. coli.

An important measure of how well we can generate inducible promoters is if they maintain the same statistics of the original dataset. Using our WGAN model, we tested TetR-induced promoters (Figure 19) using a variety of sliding mechanisms (slicing the promoter into either 3, 4, 5, or 6-mers), training the network, and finally running these results through the final node of our larger picture (Figure 3); the regression model. Strength predictions maintained the same level of correlation among these predicted strength samples and the samples generated from the GAN ($p<0.05$).

Bacterial genome pre-training—To ensure that generalizability and transfer learning is functionality with DeepInducer, we sliced a few different bacterial genomes (two E. coli genome assemblies, P. fluorescens, P. aeruginosa, and Bacillus subtilis). Our initial task was to predict whether along a longer sequence of DNA, our model could predict that a constitutive promoter existed (we later tested inducible promoters with a similar result). We compiled promoters of bp length 50-200, maintaining their position in a longer sequence, and created a training and a test set with a label of either 1 if there was a promoter within the sequence or 0 otherwise. We found that we were able to increase our MCC from 0.84 to 0.94 by using multiple genomes rather than just E. coli, as we had originally envisioned with a BERT model. Precision and recall both also increased from 0.95 and 0.89 to 0.96 and 0.96, respectively. This model was further used to generate inducible promoters in the classic text-generation route [66].

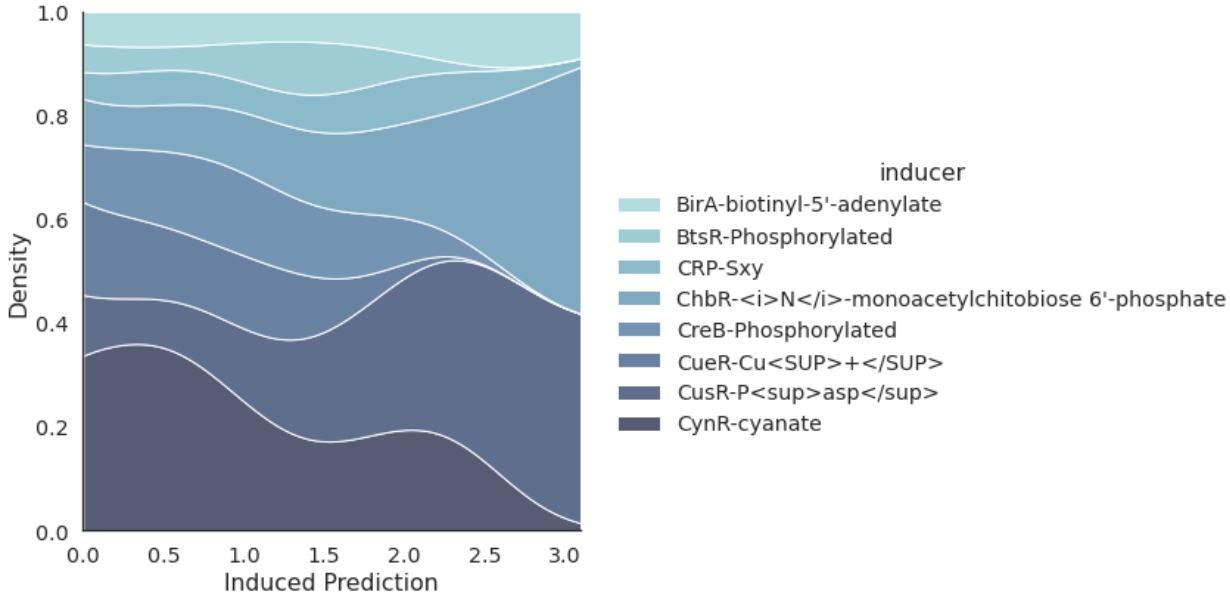


Figure 20. KDE estimation of a few generated inducible promoters based on varying -10/-35 elements and transcription factors. A higher spanning density per an inducer indicates that that ultimate inducible promoter is more likely to be expressive within that induced prediction range (where RNA expression here is normalized).

5.0 CONCLUSIONS

Given the power of modular, programmable riboregulators for diverse design applications, there is a compelling need to better integrate computational and experimental approaches. We aimed to address this prediction and design bottleneck by building STORM and NuSpeak, BioSeqML, and DeepInducer, deep learning frameworks that allow for the characterization, interpretation, and optimization of sequence data. We provide the trained models and frameworks with which to interrogate them on GitHub, constituting an accessible resource for synthetic biologists.

To take advantage of a potential boost in accuracy achieved by hybrid CNN-RNN architectures [33], we developed two complementary models. We built a CNN to function as a "motif detector" for toeholds, with several potential benefits such as the potential to capitalize on local underlying structure to train with fewer examples. However, CNNs can be limited to local sequence interactions in that they only consider a fixed number of base pairs in a sliding window, with length defined by the hyperparameters of the model. With the recent successes of language models in protein modeling [46] and promoter strength prediction, as well as the development of natural language processing techniques to elucidate RNA secondary structure [47], we implemented a language-based LSTM to further elucidate toehold sensor properties. As LSTMs consider the entire biological sequence, these model architectures are well poised to learn long-range interactions in RNA structure. In addition, while the one-hot encoding used by the CNN treats each nucleotide as an independent feature, the tokenization used as input to the LSTM implemented in our work organizes the input data into three-mers, which more closely map to codons. By using this NLP-based model in concert with convolutional architectures, we were able to reinforce the conclusions learned by each architecture.

Furthermore, we highlighted the utility of employing transfer learning techniques that use large datasets to seed models with a set of learned weights and then fine-tune on smaller, context-specific datasets to further refine the weights. After using data ablation studies to identify how both models perform with varying dataset sizes, we found that our models maintain high performance even when trained on an order of magnitude less data than present in the Angenent-Mari et al. dataset [32]. Inspired by recent work on transfer learning for transcriptomics, wherein biological patterns learned from larger datasets can be adapted to identify gene expression patterns in rare disease cohorts with few training examples, we took advantage of the models' flexibility to train with smaller datasets by using established transfer learning techniques so as to increase prediction accuracy in different toehold contexts (e.g., fused- vs. free-trigger). Finally, we demonstrated the practical value of this transfer-learning approach by using the fine-tuned models to identify optimal sensors for the SARS-CoV-2 genome (Figure 6g, h).

As the switch-trigger fusion used to generate a large toehold dataset (Figure 1a) has fundamentally different effective concentrations and stoichiometries of the trigger and switch as compared to its free-trigger counterpart, we hypothesize that our fine-tuning step is important in achieving generalizability for both models. The differences between experimental setups, including altered concentrations of chaperones, ribosomes, and transcriptional cofactors, were bridged by recycling model weights. As demonstrated in our language model training, an in silico corpus is particularly useful for training stable word and sentence embeddings. This advantage can be especially pronounced in instances where the amount of training data is sparse

or not uniformly sampled from the sequence space, which is often the case for nucleic acid datasets. These two transfer learning techniques of recycling weights and building an in silico corpus may offer an exciting avenue to translate results between experimental setups, especially in synthetic biology studies, which often have small or sparse datasets. Such approaches may be useful for enhancing design and biomining efforts to find synthetic biology parts based on only a few context-specific examples, augmented with recycled information from a larger dataset of related components.

In addition, we introduced two sequence optimization pipelines for divergent purposes. While the consensus model NuSpeak pipeline maintains base pairing complementarity and offers utility for pathogen sensor development, the STORM framework can be used to optimize toehold sequences for any performance constraints. Though gradient ascent is not a new concept, the application of generative models to redesign linear sequences for the end goal of improving function has been recently gaining traction in protein engineering [50]. For instance, generative adversarial networks [49] (GANs)—a modeling paradigm that simultaneously trains two competing neural networks—are being used to teach networks to produce realistic protein structure maps [51]. However, GANs remain challenging to train and define for biological tasks. By comparison, STORM readily converts our existing predictive CNN into a generative design tool without extensive re-training. In applications beyond engineered riboregulators, STORM could be used as a guide for nucleic acid modeling problems such as combinatorial biological circuit design, as well as to look inside of and augment existing prediction frameworks.

Next, with BioSeqML, end-users and researchers with limited computational biology knowledge can analyze and apply machine learning to their curated data. BioSeqML is a powerful code pipeline that integrates various auto-machine learning techniques and packages capable of automatically doing architecture and feature searches on nucleic acid sequences to regress or classify on a desired output. The results are very promising where for example this general system is currently able to find models with >0.85 AUC for classification of, in our case, CcaCas13b data. We believe this is significant as you can derive much of the data science intuition simply from uploading a .csv through our program, but this in turn was achieved in a fully automated fashion without any real input from the user (architecture definition or otherwise) in less than 4 hours total runtime. As part of the report we are building, we are testing this system on other larger synthetic and smaller experimental sequence databases with similar problem definitions to report.

BioSeqML's versatility is also note-worthy as it can handle various sequence data (glycans, RNA, DNA, peptides, and more), where we have not come across a tool that provides consistently high performance across many of these sequence domains. We believe this system could provide a powerful future direction of design tools for synthetic biologists with minimal computational experience. Apart from classification the pipeline also searches and generates optimized regression models, however it appears that the computational techniques and algorithms that BioSeqML is based on do have further progress that could be made, as BioSeqML is only as good as the underlying tools (in this case; DeepSwarm, AutoKeras, and TPOT). For example, on the same CcaCas13b regression data we get R^2 of around 0.2-0.55 which could be improved in future iterations. Although, this may be just a reflection of the harder problem definition and the lack of larger datasets.

Finally, with DeepInducer, end-users and researchers can apply transfer learning techniques to design inducible promoters for a multitude of organisms as well as to select strong promoters with low leakiness, or more generally, select promoters according to their specifications.

DeepInducer has shown that GFP expression can sufficiently be predicted solely on a promoter sequence, even with largely varying nucleotide positions according to various transcription factor binding sites. We built a vanilla regression model consisting of only two layers that can predict GFP expression when there is an inducer in the environment, non-induced expression when that selective inducer is not present, and the fold-change ratio between the two. What is interesting to note is that a simple model with a limited number of parameters (on the order of a few hundreds to a few thousands) can perform well, and thus can predict expression levels quickly in inference-mode (by sending a novel sample through the forward pass of the network). Although fold-change can be directly computed using both induction and non-induction values, we included this to act as a self-validating additional parameter.

We also have shown that by compiling results across literature with regards to what the inducible promoter is built up of, we can use these notions to inform an intelligent design of inducible promoters. By using specific transcription factors as the mechanism for certain inducers to react with, and -10/-35 elements as the mechanism for strength prediction, we may influence the design and overall functionality of inducible promoters. Furthermore, by mutating base pairs in certain positions (such as in the -10/-35 sequences) we can influence the level of strength by decreasing or increasing it, generally up to a ceiling of the optimal sequences as shown throughout literature.

Additionally, DeepInducer can reliably generate novel samples of DNA sequences, and most importantly, of inducible promoters. Currently, there is a GAN (WGAN) and NLP (AWD-LSTM) component that work in tandem to generate novel samples. The GAN does not produce samples conditionally, but rather based on the entire training corpus. The NLP model does produce samples conditionally [67], essentially by using surrounding nucleotides as context (given a certain transcription factor, -10 sequence, -35 sequence, etc.) and attempting to ‘fill-in-the-blank’. One route that could be explored is to add a conditional component to the GAN which would allow for a desired label (inducer) to be passed, and sequences of that label would be generated. Currently, results have not been tested experimentally but match the statistics of publicly available datasets such as described in section 4.3.

6.0 WORKS CITED

[1] Camacho, D. M., Collins, K. M., Powers, R. K., Costello, J. C. & Collins, J. J. Next-generation machine learning for biological networks. Cell 173, 1581–1592 (2018).

[2] Hallberg, Z. F., Su, Y., Kitto, R. Z. & Hammond, M. C. Engineering and in vivo applications of riboswitches. Annu. Rev. Biochem. 86, 515–539 (2017).

[3] Serganov, A. & Nudler, E. A decade of riboswitches. Cell 152, 17–24 (2013).

[4] Callura, J. M., Dwyer, D. J., Isaacs, F. J., Cantor, C. R. & Collins, J. J. Tracking, tuning, and terminating microbial physiology using synthetic riboregulators. Proc. Natl Acad. Sci. 107, 15898–15903 (2010).

[5] Rodrigo, G., Landrain, T. E. & Jaramillo, A. De novo automated design of small RNA circuits for engineering synthetic riboregulation in living cells. Proc. Natl Acad. Sci. 109, 15271–15276 (2012).

[6] Isaacs, F. J. et al. Engineered riboregulators enable post-transcriptional control of gene expression. Nat. Biotechnol. 22, 841–847 (2004).

[7] Mutalik, V. K., Qi, L., Guimaraes, J. C., Lucks, J. B. & Arkin, A. P. Rationally designed families of orthogonal RNA regulators of translation. Nat. Chem. Biol. 8, 447–454 (2012).

[8] Green, A. A., Silver, P. A., Collins, J. J. & Yin, P. Toehold switches: de-novo-designed regulators of gene expression. Cell 159, 925–939 (2014).

[9] Pardee, K. et al. Paper-based synthetic gene networks. Cell 159, 940–954 (2014).

[10] Pardee, K. et al. Rapid, low-cost detection of zika virus using programmable biomolecular components. Cell 165, 1255–1266 (2016).

[11] Ma, D., Shen, L., Wu, K., Diehnelt, C. W. & Green, A. A. Low-cost detection of norovirus using paper-based cell-free systems and synbody-based viral enrichment. Synth. Biol. 3, ysy018 (2018).

[12] Takahashi, M. K. & Lucks, J. B. A modular strategy for engineering orthogonal chimeric RNA transcription regulators. Nucleic Acids Res. 41, 7577–7588 (2013).

[13] Hutter, F., Kotthoff, L. and Vanschoren, J., Automated Machine Learning. Springer (2019).

[14] He, X., Zhao, K. and Chu, X. AutoML: A Survey of the State-of-the-Art. arXiv preprint arXiv:1908.00709 (2019).

[15] Elshawi, R., Maher, M. and Sakr, S. Automated machine learning: State-of-the-art and open challenges. arXiv preprint arXiv:1906.02287 (2019).

[16] Lipton, Z.C., Berkowitz, J. and Elkan, C. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019 (2015).

[17] Rawat, W. and Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29, 2352-2449 (2017).

[18] Zoph, B., Vasudevan, V., Shlens, J. and Le, Q.V., Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697-8710 (2018).

[19] Balaji, A. and Allen, A. Benchmarking automatic machine learning frameworks. arXiv preprint arXiv:1808.06492 (2018) .

[20] Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B. and Vanschoren, J. An open source AutoML benchmark. arXiv preprint arXiv:1907.00909 (2019).

[21] Feurer, M. and Hutter, F., ICML AutoML workshop, pp. 13 (2018).

[22] Zhang, Z., Zhou, L., Gou, L. and Wu, Y.N. Neural Architecture Search for Joint Optimization of Predictive Power and Biological Knowledge. arXiv preprint arXiv:1909.00337 (2019).

[23] Liu, D., Xu, C., He, W., Xu, Z., Fu, W., Zhang, L., Yang, J., Peng, G., Han, D. and Bai, X. AutoGenome: An AutoML Tool for Genomic Research. bioRxiv, 842526 (2019).

[24] Olson, R.S. and Moore, J.H., Automated Machine Learning. Springer, pp. 151-160 (2019).

[25] Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K. and Miikkulainen, R., Proceedings of the Genetic and Evolutionary Computation Conference, pp. 401-409 (2019).

[26] Li, S.Y., Cheng, Q.X., Liu, J.K., Nie, X.Q., Zhao, G.P. and Wang, J. CRISPR-Cas12a has both cis- and trans-cleavage activities on single-stranded DNA. Cell Research, 28, 491-493 (2018).

[27] Gootenberg, J.S., Abudayyeh, O.O., Lee, J.W., Essletzbichler, P., Dy, A.J., Joung, J., Verdine, V., Donghia, N., Daringer, N.M. and Freije, C.A. Nucleic acid detection with CRISPR-Cas13a/C2c2. Science, 356, 438-442 (2017).

[28] Yu, T.C., Liu, W.L., Brinck, M.S. et al. Multiplexed characterization of rationally designed promoter architectures deconstructs combinatorial logic for IPTG-inducible systems. Nat Commun 12, 325 (2021).

[29] Chen, Y., Ho, J.M.L., Shis, D.L. et al. Tuning the dynamic range of bacterial promoters regulated by ligand-inducible transcription factors. Nat Commun 9, 64 (2018).

[30] Pedone, E., Postiglione, L., Aulicino, F. et al. A tunable dual-input system for on-demand dynamic gene expression regulation. Nat Commun 10, 4481 (2019).

[31] Gama-Castro, Socorro; Salgado, Heladia; Santos-Zavaleta, Alberto; Ledezma-Tejeida, Daniela; Muñiz-Rascado, Luis; García-Sotelo, Jair Santiago; Alquicira-Hernández, Kevin; Martínez-Flores, Irma; Pannier, Lucia. RegulonDB version 9.0: high-level integration of gene regulation, coexpression, motif clustering and beyond. Nucleic Acids Research. 44 (2016).

[32] Angenent-Mari, N. M., Garruss, A. S., Soenksen, L. R., Church, G. & Collins, J. J. A deep learning approach to programmable RNA switches. Nat. Commun. https://doi.org/10.1038/s41467-020-18677-1 (2020).

[33] Zeiler, M. D. & Fergus, R. Visualizing and understanding convolutional networks. Computer Vis. – ECCV 2014 8689, 818–833 (2014).

[34] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015).

[35] Abadi, M. et al. TensorFlow: large-scale machine learning on heterogeneous distributed systems. Preprint at https://arxiv.org/abs/1603.04467 (2016).

[36] Howard, J. & Ruder, S. Universal language model fine-tuning for text classification. ACL 1, 328–339 (2018).

[37] Bradbury, J., Merity, S., Xiong, C. & Socher, R. Quasi-recurrent neural networks. ICLR 2017, 1–12 (2016).

[38] Loshchilov, I. & Hutter, F. SGDR: stochastic gradient descent with warm restarts. ICLR 2017, 1–16 (2017).

[39] Müller, R., Kornblith, S. & Hinton, G. E. When does label smoothing help?. Adv. Neural Inform. Process. Syst. 32, 4694–4703 (2019).

[40] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. Distributed representations of words and phrases and their compositionality. Adv. Neural Inform. Process. Syst. 26, 3111–3119 (2013).

[41] Bogard, N., Linder, J., Rosenberg, A. B. & Seelig, G. A deep neural network for predicting and engineering alternative polyadenylation. Cell 178, 91–106.e23 (2019).

[42] Zadeh, J. N., Wolfe, B. R. & Pierce, N. A. Nucleic acid sequence design via efficient ensemble defect optimization. J. Comput. Chem. 32, 439–452 (2011).

[43] Cuperus, J. T. et al. Deep learning of the regulatory grammar of yeast 5′ untranslated regions from 500,000 random sequences. Genome Res. 27, 2015–2024 (2017).

[44] McInnes, L., Healy, J., Saul, N. & Großberger, L. UMAP: uniform manifold approximation and projection. J. Open Source Softw. 3, 861 (2018)

[45] Lu, R. et al. Genomic characterisation and epidemiology of 2019 novel coronavirus: implications for virus origins and receptor binding. Lancet 395, 565–574 (2020).

[46] Biswas, S., Khimulya, G., Alley, E. C., Esvelt, K. M. & Church, G. M. Low-N protein engineering with data-efficient deep learning. biorxiv Preprint at https://www.biorxiv.org/content/10.1101/2020.01.23.917682v2 (2020).

[47] Yonemoto, H., Asai, K. & Hamada, M. A semi-supervised learning approach for RNA secondary structure prediction. Comput. Biol. Chem. 57, 72–79 (2015).

[48] Simonyan, K., Vedaldi, A. & Zisserman, A. Deep inside convolutional networks: visualising image classification models and saliency maps. ICLR 2014, 1–8 (2014).

[49] Goodfellow, I. et al. Generative adversarial nets. Proc. Adv. Neural Inform. 27, 2672–2680 (2014).

[50] Yang, K. K., Wu, Z. & Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. Nat. Methods 16, 687–694 (2019).

[51] Anand, N. & Huang, P. Generative modeling for protein structures. Proc. Adv. Neural Inform. Process. Syst. 31, 7494–7505 (2018).

[52] Jin, H., Song, Q. and Hu, X., Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, pp. 1946-1956 (2019).

[53] Byla, E. and Pang, W. DeepSwarm: Optimising Convolutional Neural Networks using Swarm Intelligence. arXiv preprint arXiv:1905.07350 (2019).

[54] Olson, R.S. and Moore, J.H., Automated Machine Learning. Springer, pp. 151-160 (2019).

[55] Le, T.T., Fu, W. and Moore, J.H. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. Bioinformatics, 36, 250-256 (2020).

[56] Knott, G.J. and Doudna, J.A. CRISPR-Cas guides the future of genetic engineering. Science, 361, 866-869 (2018).

[57] Zetsche, B., Gootenberg, J.S., Abudayyeh, O.O., Slaymaker, I.M., Makarova, K.S., Essletzbichler, P., Volz, S.E., Joung, J., Van Der Oost, J. and Regev, A. Cpf1 is a single RNA-guided endonuclease of a class 2 CRISPR-Cas system. Cell, 163, 759-771 (2015).

[58] Abudayyeh, O.O., Gootenberg, J.S., Konermann, S., Joung, J., Slaymaker, I.M., Cox, D.B., Shmakov, S., Makarova, K.S., Semenova, E. and Minakhin, L. C2c2 is a single-component programmable RNA-guided RNA-targeting CRISPR effector. Science, 353 (2016).

[59] Gootenberg, J.S., Abudayyeh, O.O., Kellner, M.J., Joung, J., Collins, J.J. and Zhang, F. Multiplexed and portable nucleic acid detection platform with Cas13, Cas12a, and Csm6. Science, 360, 439-444 (2018).

[60] Chow CN, Chiang-Hsieh YF, Chien CH, et al. Delineation of condition specific Cis- and Trans-acting elements in plant promoters under various Endo- and exogenous stimuli. BMC Genomics. 2018;19(Suppl 2):85. (2018).

[61] Cooper GM. The Cell: A Molecular Approach. 2nd edition. Sunderland (MA): Sinauer Associates; (2000). https://www.ncbi.nlm.nih.gov/books/NBK9850/.

[62] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre- training of deep bidirectional transformers for language under- standing. arXiv preprint arXiv:1810.04805 (2018).

[63] Merity, S., Shirish Keskar, N., and Socher, R. Regularizing and Optimizing LSTM Language Models. arXiv preprint arXiv:1708.02182 (2017).

[64] Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein Generative Adversarial Networks. Proceedings of Machine Learning Research (2017). https://proceedings.mlr.press/v70/arjovsky17a.html.

[65] Rácz, A., Bajusz, D. & Héberger, K. Life beyond the Tanimoto coefficient: similarity measures for interaction fingerprints. *J Cheminform* **10,** 48 (2018). https://doi.org/10.1186/s13321-018-0302-y.

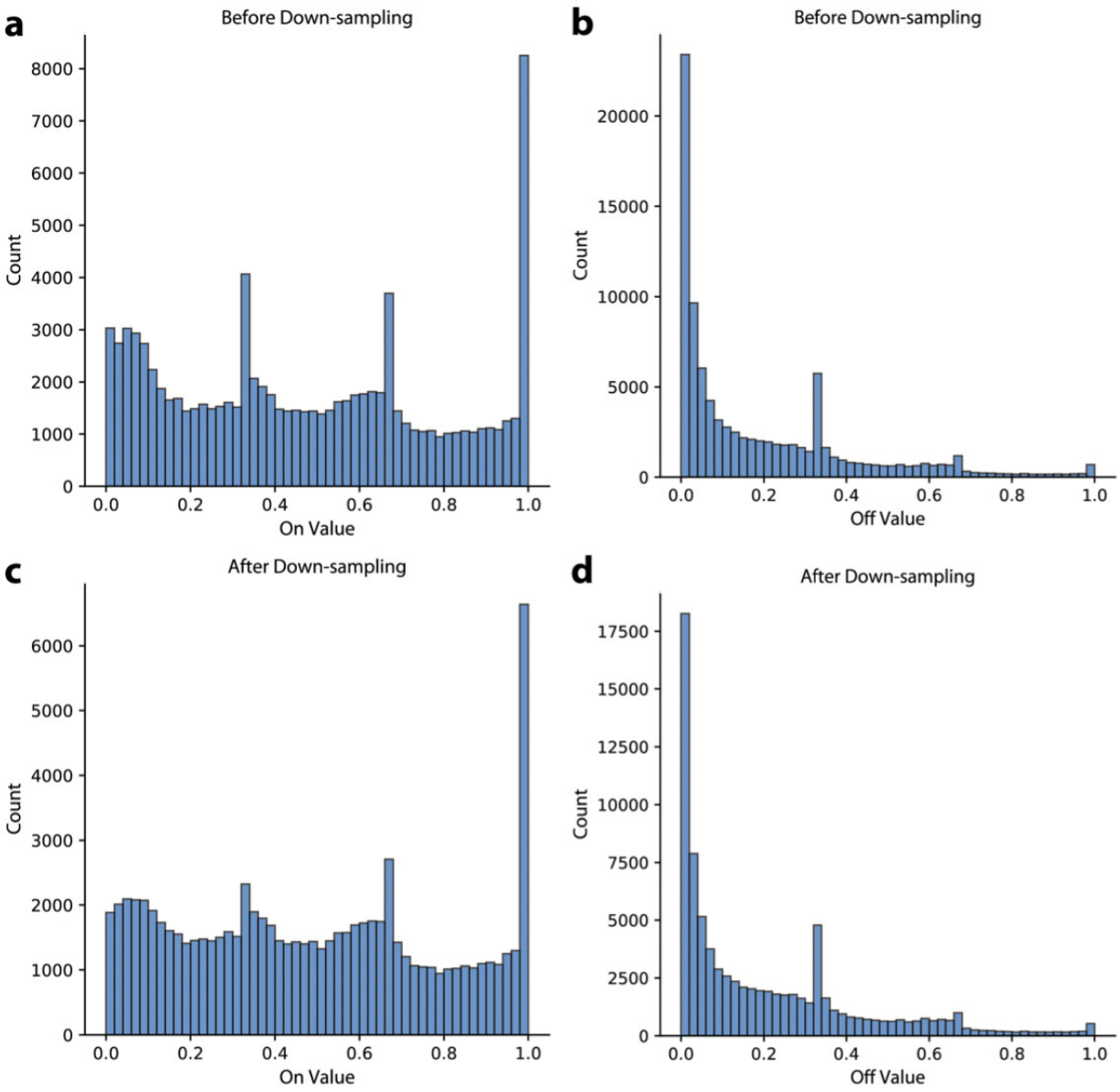
[66] Touseef Iqbal, Shaima Qureshi, The survey: Text generation models in deep learning, Journal of King Saud University - Computer and Information Science, ISSN 1319-1578, (2020) https://doi.org/10.1016/j.jksuci.2020.04.001.

[67] Mirza, Mehdi, and Simon Osindero. "Conditional Generative Adversarial Nets." *ArXiv:1411.1784 [Cs, Stat]*, (2014). http://arxiv.org/abs/1411.1784.

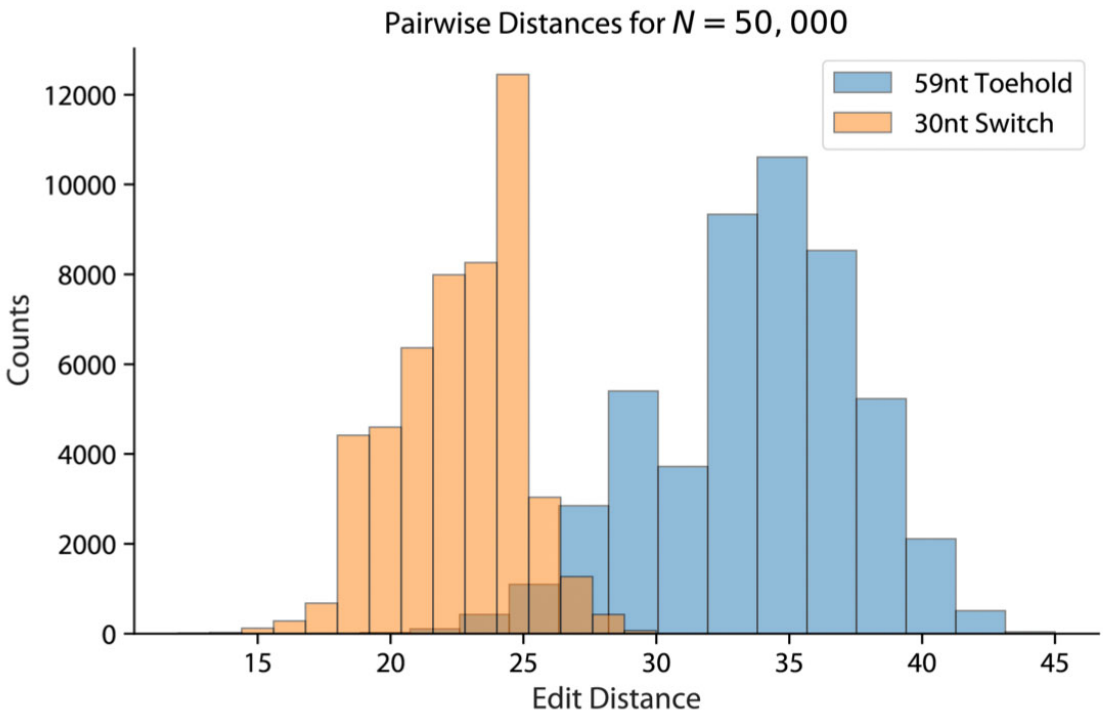
[68] Surowiecki, James. The Wisdom of Crowds. New York: Anchor Books, 2005. Print.

[69] Faith JJ, Hayete B, Thaden JT, Mogno I, Wierzbowski J, Cottarel G, et al. Large-Scale Mapping and Validation of Escherichia coli Transcriptional Regulation from a Compendium of Expression Profiles. PLoS Biol 5(1): e8. (2007) https://doi.org/10.1371/journal.pbio.0050008.

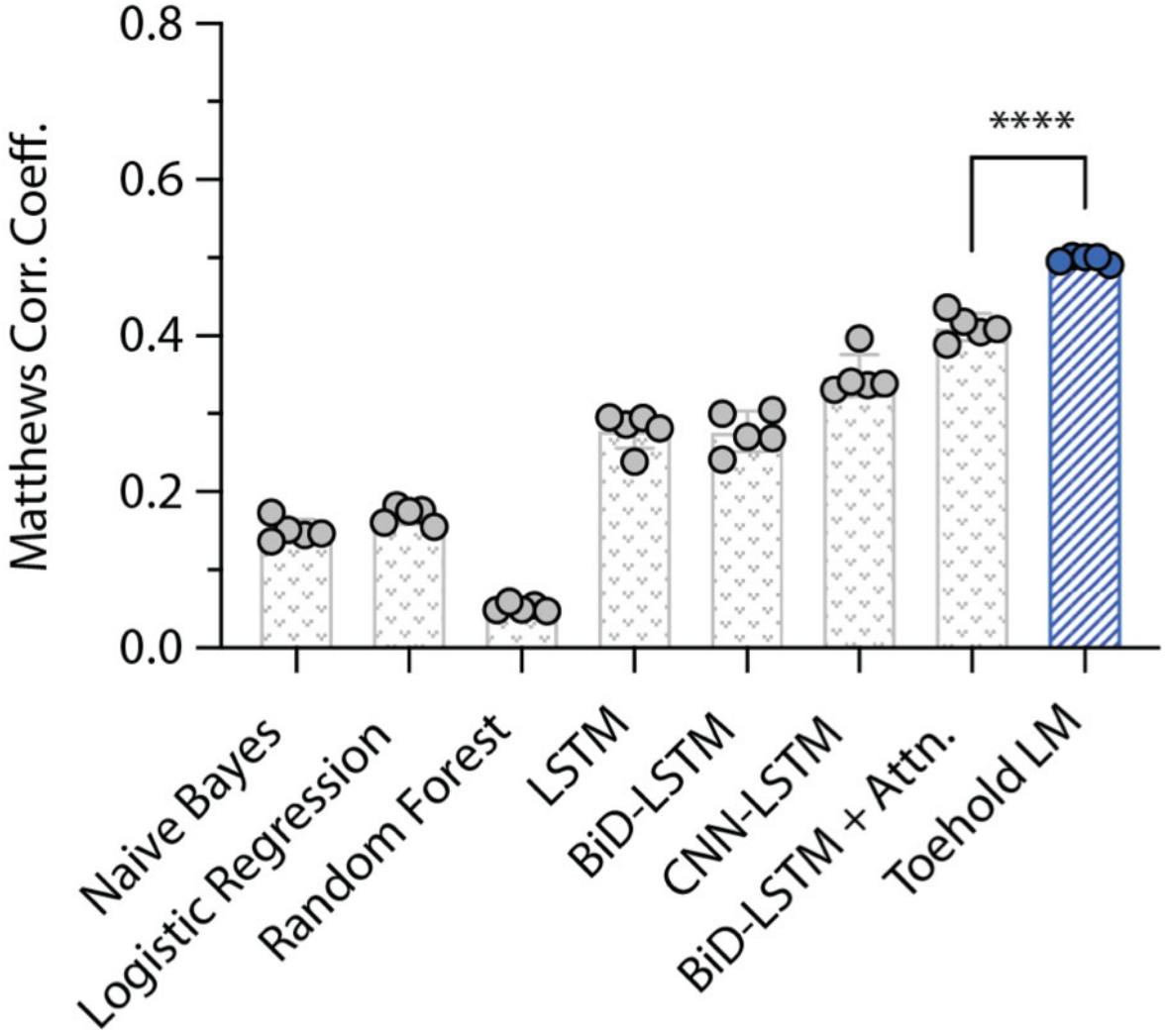
APPENDIX



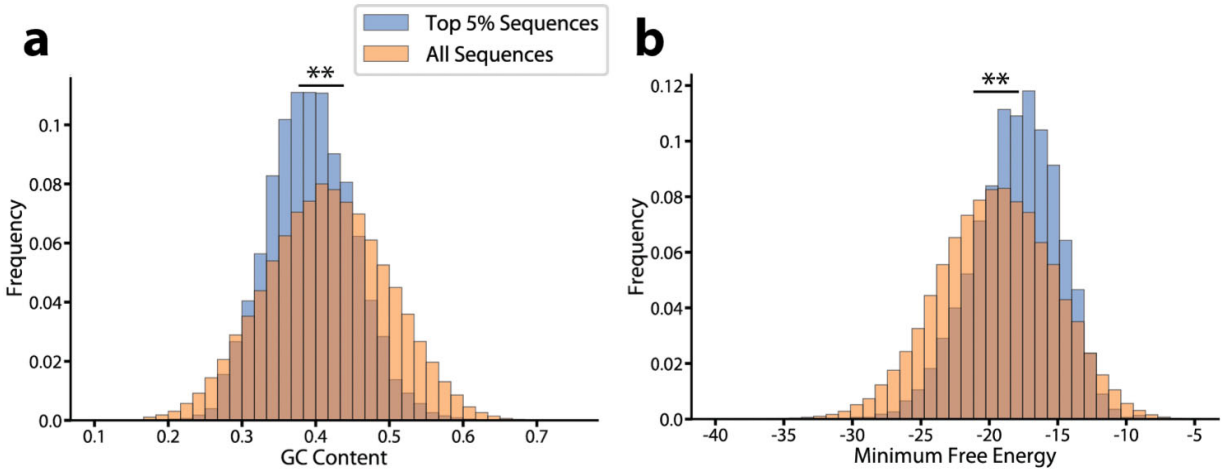
Supplementary Figure S1. Down-sampling preserves the ON and OFF value distributions, while reducing experimental artifacts. a) Given the abnormally high counts observed at several ON and b) OFF values, the ON and OFF distributions were trimmed such that the number of sequences in each of 1000 evenly-spaced bins was reduced to the mean number of sequences over all bins. We took the union of the sequences that passed either ON or OFF filtering for a total of 81,155 sequences, showing here the resulting c) ON and d) OFF value distributions.



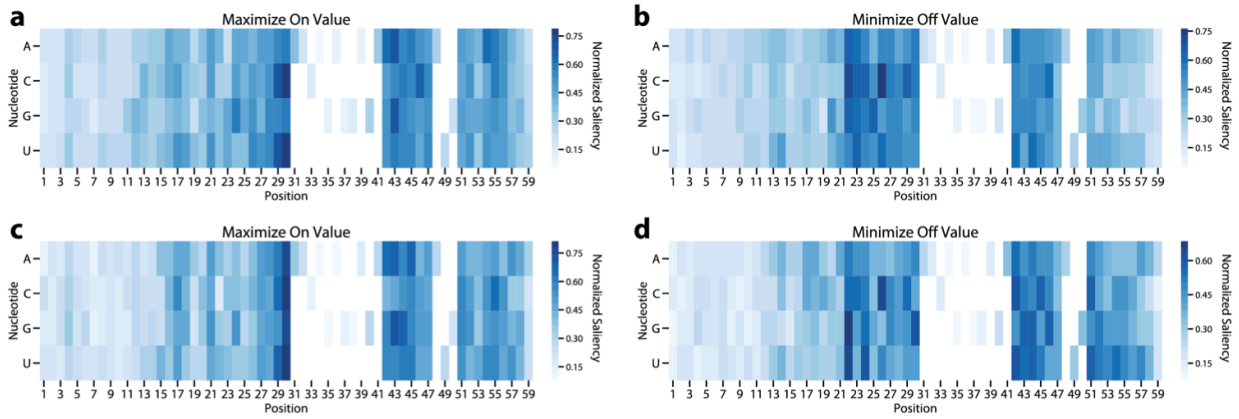
Supplementary Figure S2. Pairwise edit distances between sets of random toeholds show little collinearity in synthetic sequences. To demonstrate that the synthetically generated toeholds were sufficiently different from each other, we calculated pairwise edit distances for ten sets of 5,000 toeholds for both the 30-nucleotide variable switch region (orange) and the corresponding 59 nucleotide toehold (blue). The peak for both distributions in the case of highly similar sequences would be close to 0. These distributions are upper bounded at 30 nucleotides and 45 nucleotides for the switch and toehold, respectively, due to conserved regions in the toehold. We observe the prevalence of edit distances between 23 and 25 nucleotides for the switch sequences, and between 35 and 40 for the toeholds, which reinforce the expectation of dissimilarity between sequences and consequent non-collinearity.



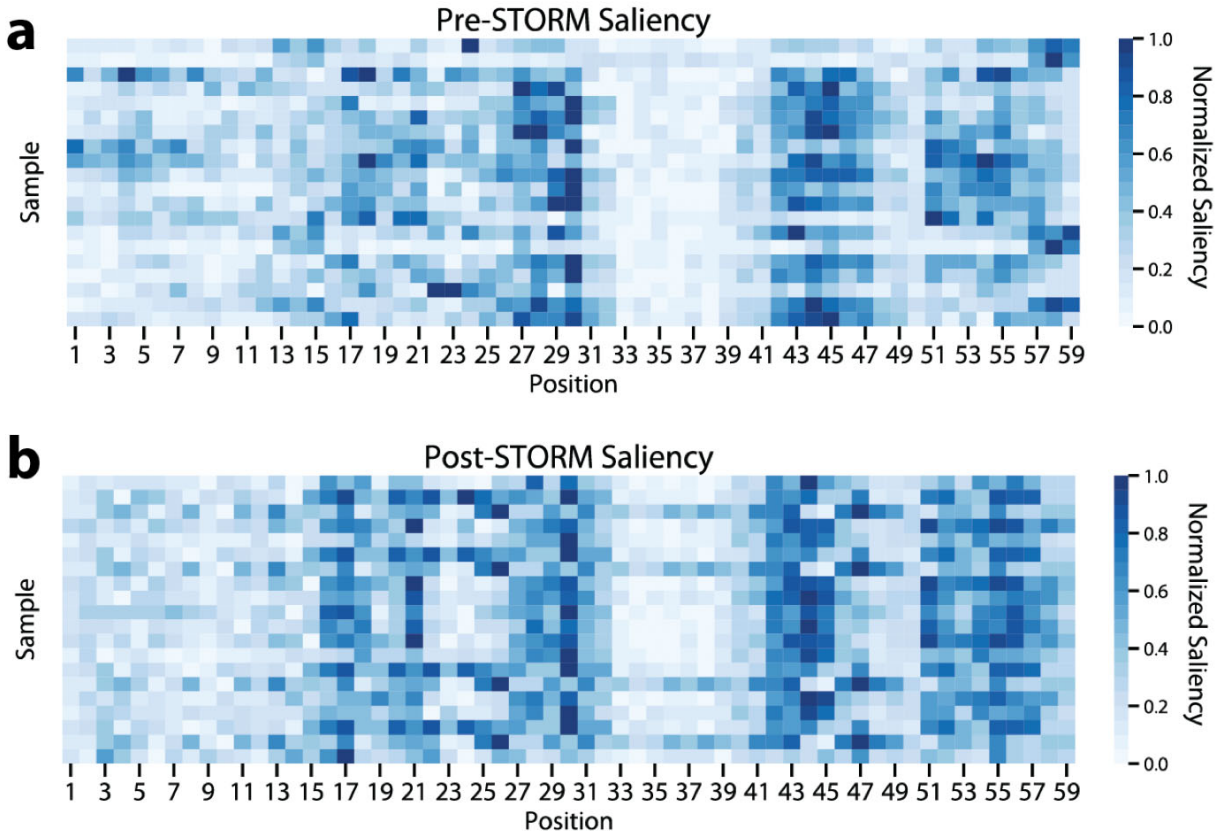
Supplementary Figure S3. Language model-based classification outperforms off-the-shelf classification models. We compared our language model (LM) to several off-the-shelf classification models, each in a five-fold cross validation setting. The toehold language model significantly improved upon the bidirectional LSTM (Long Short-Term Memory) model with an attention layer when classification was evaluated with Matthews Correlation Coefficient ($p = 5.01x10\text{-}6$). Error bars represent mean +/- standard deviation for N = 5 cross validation folds and all tests are two-tailed t-tests.



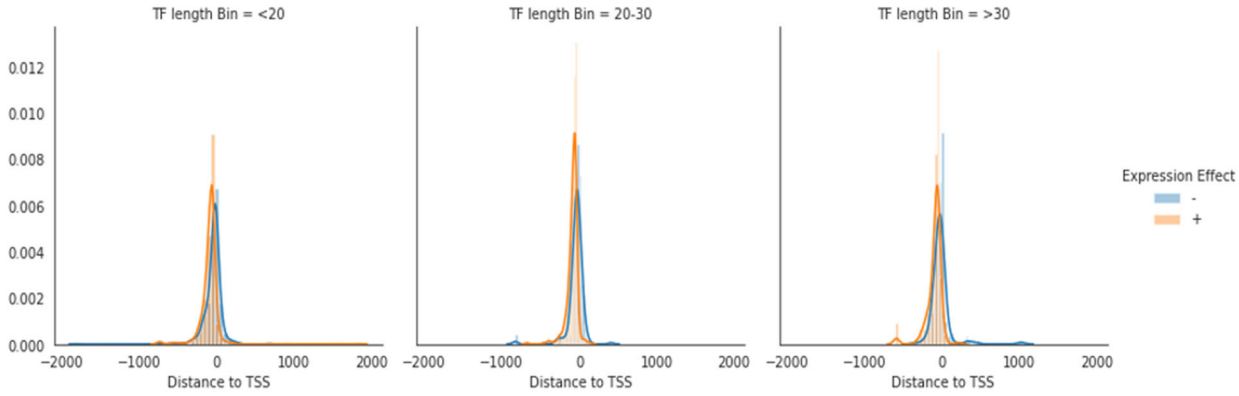
Supplementary Figure S4. Biophysical properties are not adequate to predict toehold performance on their own. a) GC content distributions and b) switch minimum free energy estimates according to NUPACK were calculated for all sequences (orange, N = 91,534) and the top 5% of sequences (blue, N = 4,577), with clear differences in distributions for GC content (p = 1.21x10-87) and minimum free energy (p = 2.63x10-132). However, given the large overlaps in distributions, these properties are not solely predictive of toehold performance on their own. All tests are two-tailed Mann-Whitney U tests.



Supplementary Figure S5. Saliency maps are not markedly different for toehold switches with good and bad performance. a) To ensure that the saliency maps did not change dramatically based on which sequences were tested, saliency was calculated on a set of 100 'good' switches using the convolutional neural network model. Saliency maps were generated to both maximize the ON prediction and b) minimize the OFF prediction. c) Saliency maps were also generated for 100 'bad' switches, both maximizing the ON prediction and d) minimizing the OFF prediction. Switches were randomly chosen from top and bottom performing toeholds based on experimental ON/OFF ratios. All saliency maps indicate the importance of the regions immediately surrounding the Shine-Dalgarno sequence, while de-emphasizing the importance of the first 12 nucleotides.



Supplementary Figure S6. Differential attention to first twelve nucleotides in saliency maps for gradient ascent sequences pre- and post-optimization. a) Given a set of 20 bad toeholds that were optimized via our gradient ascent framework, we sought to understand how the model's focus changed when predicting performance before and after gradient ascent. Saliency maps to illustrate the parts of the sequence that maximized the ON prediction were generated for the pre-optimization sequences and b) post-optimization sequences. For several of the 20 sequences sampled, the model pays less attention to the first 12 nucleotides after optimization.



Supplementary Figure S7. Transcription factors are binned by their length in nucleotides and measured via their distance to their respective transcription start site (TSS), and colored by their expression effect (- is repressed and + is activated) regarding influence by an inducer (normalized by total TF count).

List of Symbols

The following list describes all symbols used in this technical report.

Term	Definition
µL	Microliter
nM	Nanometer
GHz	Gigahertz

List of Abbreviations and Acronyms

The following list describes all abbreviations and acronyms used in this technical report.

Term	Definition
RNA	Ribonucleic Acid
AA	Amino Acid
AGSD-LSTM	Average SGD Weight-Dropped LSTM
AUC	Area Under Curve
auROC	area under the Receiver Operating Characteristic
BERT	Bidirectional Encoder Representations from Transformers
BLAST	Basic Local Alignment Search Tool
CGR	Chaos-game Representations
CNN	Convolutional Neural Network
DL	Deep Learning
DNA	Deoxyribonucleic Acid
GAN	Generative Adversarial Network
GFP	Green Fluorescent Protein
GPU	Graphics Processing Unit
IPTG	Isopropyl β-d-1-thiogalactopyranoside
LM	Language Model
LSTM	Long Short-Term Memory
ML	Machine learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
N	Sample Size
NLP	Natural Language Processing
NUA	Nuclear Pore Anchor
NUPACK	Nucleic Acid Package
PCR	Polymerase Chain Reaction
QRNN	Quasi-Recurrent Neural Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operator Characteristic
R^2	Coefficient of Determination
SD	Shine–Dalgarno
SD2	Synergistic Discovery and Design
S.E.M.	Standard Error of the Mean

SGD	Stochastic Gradient Descent
TF	Transcription Factor
TSS	Transcription Start Site
UMAP	Uniform Manifold Approximation and Projection for Dimension Reduction
	Chemistry-Concentration
v/v	