Data-Driven Protein Engineering

Thesis by
Zachary Wu

In Partial Fulfillment of the Requirements for the
Degree of
Chemical Engineering

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2021
Defended June 25, 2020

Zachary Wu
ORCID: 0000-0003-2429-9812

To my parents, Tsui-Feng Chang and Chang-Nien Wu, for their love and nurturing

and to Andrea Wu, my sister and life-long best friend

"*All models are wrong, but some are useful.*"

George E. P. Box

ACKNOWLEDGEMENTS

Caltech has the smallest campus I've ever seen. The annual 5K run actually requires *two* laps around campus, which I discovered after spending all my energy on the first. Despite its compactness, it's quite easy to get lost in the sheer quantity of exciting research, especially considering the passion of those conducting it.

Therefore, I must thank first and foremost my advisor, Frances Arnold, for guiding me on this journey. Frances has an uncanny ability to discern not only what is interesting research, but *useful* as well, and that is only one angle by which she is able to challenge and sharpen ideas and their implementation. I am deeply grateful for both the freedom she has given to me and the (mostly) gentle direction needed to put it to good use.

Caltech is filled with numerous instructors to guide the students and research. I am particularly grateful for Yisong Yue, who is a patient and insightful teacher that has guided me both in all of my machine learning coursework and in project design. My meetings with Yisong have become less regular over the years, but each provided ample guidance for months. Zhen-Gang Wang and Tom Miller gave me an appreciation for statistical thermodynamics that enabled a smoother transition to machine learning, and Justin Bois taught me how to think carefully about experimental data. Justin Bois and Mike Vicic are two of the most passionate individuals I've met in mentoring students, and they are my role models for teaching. Caltech is much brighter with them leading. I am also grateful for the chair of my thesis committee, David Tirrell, who has a sharp insight with breathtaking scope that is always ready to challenge and improve ideas in any field.

I thank Amazon Web Services, BASF, the NIH, and the NSF for funding various portions of my projects. I am particularly grateful to Alex Ford and Michael Liszka for their irreplaceable help in collaborations with the first two companies, and to Kimberly Mayer for showing me how to communicate science effectively in writing.

The Arnold group is an excellent place to meet some of the most talented and caring people, and I am immensely thankful to Oliver Brandenberg, my rotation mentor, for getting me started. I am grateful to have overlapped with many members of the Arnold group, but particularly thankful to have met Grzegorz Kubik, Rusty Lewis, Jennifer Kan, David Romney, Tina and Brad Boville, and Xiongyi Huang. Of course, Sabine Brinkmann-Chen and Nat Goldberg, who keep us all happy and safe (an often

thankless job), deserve special mention. I am also incredibly grateful for finding a crew that I could always count on in Samuel Ho, Kari Hernandez, and Kelly Zhang. I'm happy that we're still navigating adulthood together even after you've all left me in Pasadena. Kelly deserves special mention, as she has challenged me and supported me in ways I didn't know I needed.

The Arnold group has a long history of convincing proteins to improve with machine learning. I was very fortunate to spend most of my graduate career with Kevin Yang. I couldn't ask for a better colleague and friend, and this field is much cozier knowing he's a part of it. Sometimes I forget he's graduated given the amount we keep up, but there's so many interesting things coming out to talk about! Within our growing machine learning subgroup, I am particularly grateful to have worked with Alycia Lee, Bruce Wittmann, and Kadina Johnston. I hope I've been able to give back a fraction of what you've taught me.

I must also thank the communities I was so fortunate to be a part of in Pasadena, including pickup basketball, chamber music, and Epicentre. I will miss you all dearly.

Most importantly, I thank my family for their never-ending support. My parents and sister have showered me with unconditional love in ways that still amaze me. It's been a crazy journey since I left our cozy town, but my home will always be with you.

ABSTRACT

Directed evolution has enabled the adaptation of natural protein sequences for an endless variety of human applications. Given a starting point — a sequence with measurable activity — directed evolution is able to improve protein sequences by iteratively accumulating beneficial mutations. However, directed evolution requires investing large experimental effort, which continues to be the major bottleneck in efficient protein optimization. To this end, we describe a framework for incorporating machine learning in the directed evolution process to maximize the utility of generated experimental data in Chapter 2. In Chapter 3, we then show that this framework outperforms traditional directed evolution methods on an empirical fitness landscape. However, directed evolution is fundamentally limited by its need for a starting point, or a sequence with measurable activity. To tackle this issue, we test the ability of nascent deep learning techniques for generating short, functional amino acid sequences in Chapter 4. Encouraged by this success, we attempted to generate full length enzymatic sequences for desired substrates without success. However, we were able to apply this deep learning approach to model other aspects of enzymatic protein sequences in Chapter 5. Finally, the field of data-driven protein sequence generation is enjoying a recent surge in interest, and we provide an updated review of protein engineering with machine learning, focusing on recent work in deep generative modeling in Chapter 1.

PUBLISHED CONTENT AND CONTRIBUTIONS

1. Wu, Z. *et al.* Signal Peptides Generated by Attention-Based Neural Networks. *ACS Synthetic Biology* **9,** 2154–2161. doi:10.1021/acssynbio.0c00219 (2020).
   Z.W. conceived and directed this study. Z.W. participated in analyzing the generative model results and planning the experimental validation. Z.W. analyzed the experimental results and wrote the manuscript.

2. Wu, Z., Kan, S. B. J., Lewis, R. D., Wittmann, B. J. & Arnold, F. H. Machine learning-assisted directed protein evolution with combinatorial libraries. *Proc Natl Acad Sci USA*. doi:10.1073/pnas.1901979116 (2019).
   Z.W. conceived the study and performed all experiments, simulations, and data analysis.

3. Yang, K. K., Wu, Z. & Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nature Methods* **16,** 687–694. doi:10.1038/s41592-019-0496-6 (2019).
   Z.W. participated in the writing of this review.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

ENGINEERING PROTEIN SEQUENCES WITH DEEP LEARNING

Contributions Statement: Zachary Wu performed the literature review and wrote the initial draft. Sabine Brinkmann-Chen, Kadina Johnston, and Kevin Kaichuang Yang provided helpful suggestions and edits.

1.1 Abstract

Protein engineering seeks to identify protein sequences with optimized properties. When guided by machine learning, protein sequence generation methods can draw on prior knowledge and experimental efforts to improve this process. In this chapter, we highlight recent applications of machine learning in generation of protein sequences, focusing on the emerging field of deep generative methods. The first section, learned representations in regression models, is a logical progression to the work we present in Chapters 2 and 3. By pre-training models to learn representations on all protein sequences, regression models have reduced burden in modeling downstream protein tasks. We attempt a similar approach to co-representing proteins and small molecules in Chapter 5. The next section, generative models, discusses popular methods for functional sequence generation, including our recent work published in Chapter 4. We end this review chapter with recent research in optimizing with deep generative models, which captures the protein engineer's desire for identifying fitness optima to these emerging approaches.

1.2 Introduction

Proteins are highly-regarded as the workhorse molecules of natural life, and they are quickly being adapted for human-designed purposes as well. These macromolecules are encoded as linear chains of amino acids, which then fold into dynamic 3-dimensional structures that accomplish a staggering variety of functions. To improve proteins for human purposes and new-to-nature reactions, protein engineers have developed a variety experimental and computational methods for designing their sequences and structures [1–4]. A developing paradigm, machine learning-guided protein engineering, promises to leverage the information obtained from wet-lab experiments with data-driven models [5–7].

Much of the early work has focused on incorporating regressive models trained on

fitness measurements related to specific tasks to guide protein engineering [5]. As protein engineering is an optimization process for improving protein properties (such as selectivity or stability), other optimization methods also have strong potential. Specifically, unsupervised learning approaches are improving the protein engineering paradigm. While studies incorporating knowledge of protein structure are becoming increasingly powerful [8–11], they are beyond the scope of this review, and we focus on deep generative models of protein sequence.

We discuss three applications of deep generative models in protein engineering. In the first section, we briefly highlight the incorporation of learned protein sequence embeddings in downstream supervised engineering tasks, an important improvement to an established framework for protein engineering. In the second, we highlight examples of protein sequence generation through popular generative models. Finally, we highlight emerging examples of optimization with generative models in protein sequence design. Where possible, these methods are introduced with case studies that have validated these generated sequences *in vitro*.

1.3 Learned Representations in Regression Models

An established framework for applying machine learning to guide protein engineering is through the training and application of supervised regression models for specific tasks, which is better reviewed elsewhere [5, 6]. Early examples of this approach were developed by Fox [12] and Liao [13] in learning the relationship between cyanation and hydrolysis activity (respectively). Briefly, in this approach, sequence-function experimental data are used to train regressive models. These models are then used as estimates for the true experimental value, and can be used to search through and identify beneficial sequences *in silico*. In training these regressive models, one-hot encodings are a powerful baseline for representing protein sequences. However, embeddings learned by deep learning have the potential to provide more informative encodings.

For example, Biswas and coauthors recently applied learned representation from a Long Short-Term Memory (LSTM) network to encode protein sequences [14]. First, they train an LSTM to predict the next amino acid in a protein given the previous amino acids using sequences in UniRef50 [15]. They fine-tune this representation on evolutionarily related sequences, and then use the activations from the penultimate layer to represent each position in an input protein sequence. The authors propose that this approach is capable of generating meaningful encodings for protein sequences.

The desired advantage of this approach over directly embedding physicochemical properties [16] is that the deep representation includes contextual information, such that the downstream supervised model does not have to see examples of a position's mutations in order to make accurate predictions at that position. While this ideal may never be achieved, Biswas and coauthors show promising results when applied to two tasks: improving the fluorescence activity of *Aequorea victoria* green fluorescent protein (avGFP) and in optimizing TEM-1 β-lactamase. After training on just 24 randomly selected sequences, this approach consistently outperforms one-hot encodings with 5 to 10 times the hit rate (defined as the fraction of proposed sequences with activity greater than wild type).

1.4 Generative Models: Introduction

While discriminative models are useful in cases with fewer than thousands of labeled examples, generative models provide a powerful alternative when there is sufficient data to learn from. By modeling the distribution by which the data are generated, generative models enable direct identification of new sequences. While generative models typically require large amounts of data ($\sim 10^5$), advances in deep mutational scanning [17] and recently plate assays [18], are enabling these approaches. Here, we describe three popular generative models, variational autoencoders, generative adversarial networks, and autoregressive models, and provide examples of their applications to protein sequences. These models are summarized in Figure 1.1.

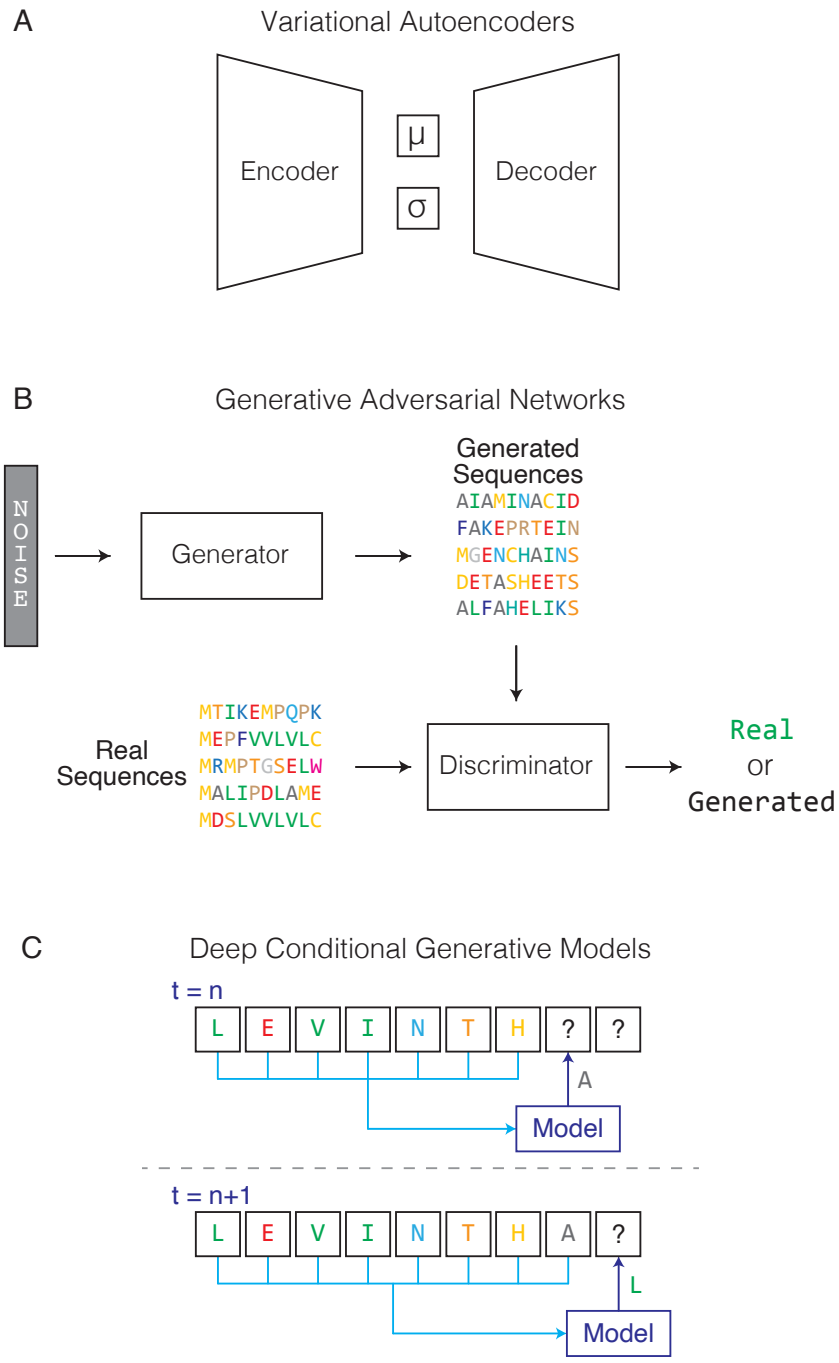


Figure 1.1: (A) Variational Autoencoders (VAEs) are tasked with encoding sequences in a structured latent space. Samples from this latent space may then be decoded to functional protein sequences. (B) Generative Adversarial Networks (GANs) have two networks locked in a Nash equilibrium: the generative network generates synthetic data that look real, while the discriminative modelvdiscerns between real and synthetic data. (C) autoregressive models predict the next amino acid in a protein given the amino-acid sequence up to that point.

Variational Autoencoders: Background

To provide an intuitive introduction to Variational Autoencoders, we first introduce the concept of autoencoders [19–21], which are comprised of an encoder and a decoder. The encoder, $q(z|x)$, maps each input x_i into a latent representation z_i. This latent representation is comparatively low dimension to the initial encoding, creating an information bottleneck that forces the autoencoder to learn a useful representation. The decoder, $p(x|z)$, reconstructs each input x_i from its latent representation z_i. During training, the goal of the model is to maximize the probability of the data $p(x)$, which can be determined by marginalizing over z:

$$p(x) = \int p(x|z)p(z)dz \tag{1.1}$$

However, direct evaluation of this integral is intractable and is instead bounded using variational inference. It can be shown that a lower bound of $p(x)$ can be written as the following [19]:

$$\log p(x) \geq \mathbb{E}\left[\log p(x|z)\right] - \mathbb{D}_{KL}\left[q(z|x)||p(z)\right] \tag{1.2}$$

where $\mathbb{D}_{KL}$ is the Kullback-Leibler divergence, which can be interpreted as a regularization term that measures the amount of lost information when using q to represent p, and the first expectation $\mathbb{E}$ term represents reconstruction accuracy. Thus, variational autoencoders aim for accurate reconstruction while maintaining structure in the latent space through regularization. Variational autoencoders impose the added constraint that the latent representation are well approximated by Gaussian normal distributions, such that $p(z) = \text{Normal}(0, 1)$. Intuitively, this constraint enables smooth interpolation between points in the latent representation, enforcing structure in an otherwise arbitrary encoding.

Variational Autoencoders: Examples

VAEs can be employed in a variety of ways. For example, after training on an alignment of similar sequences, the predicted probability of a sequence can be used as a proxy for the relative fitness of that sequence, which can correlate well with various sequence-function datasets obtained through deep mutational scans [22, 23]. Alternatively, generating a new sequence from a VAE is as simple as generating a random (Gaussian) latent vector and passing it through the decoder. This random

sequence is expected to be a non-linear interpolation of the sequences used to train the VAE.

More powerfully, as discrete protein sequences are now represented in a continuous and compressed latent space, optimization of these sequences can be reinterpreted as optimization in the latent space. Specifically, a simple downstream supervised model can be trained on protein sequence-function relationships, where the protein sequence is input as its latent encoding. In BioSeqVAE, the latent representation was learned from 200,000 sequences obtained from SwissProt between 100 and 1000 amino acids in length [24]. The authors demonstrate that a simple random forest classifier from scikit-learn [25] can be used to learn the relationship between roughly 60,000 sequences (represented by the autoencoder) and their protein localization and enzyme classification (by Enzyme Commission number). By optimizing for either property through the downstream models in latent space, and converting this latent representation to a protein sequence, the authors generate examples that have either or both desired properties. While the authors did not validate the generated proteins *in vitro*, they did observe sequence homology between their designed sequences and natural sequences with the desired properties.

Sequences are typically designed *for* a specific task, and task-specific information can be incorporated in the training process [26]. For example, the decoder can be conditioned on the identity of the metal cofactors bound [27]. After training on 145,000 enzyme examples in MetalPDB [28], the authors find a higher fraction of desired metal-binding sites observed in generated sequences. Additionally, the authors show that 11% of 1000 sequences generated for recreating a removed copper-binding site identified the correct binding amino acid triad. The authors also applied this approach to designing for specified protein folds, validating their results with Rosetta and molecular dynamics simulations.

A comparatively small body of literature has successfully tested VAE sequences with wet-lab validation, but Hawkins-Hooker and coauthors have succeeded after careful consideration of protein alignment [29]. The authors consider two methods of approaching protein sequences: 1) by computing the alignment first and training a VAE (MSA VAE) and 2) by introducing an autoregressive component to the decoder to learn the unaligned sequences (AR VAE). Motivated by a similar model used for text generation [30], the decoder of the AR VAE contains an up-sampling component, which converts the compressed representation to the length of the output sequence, and a recurrent neural network, which we discuss later. Both models were trained

with roughly 70,000 luciferase sequences (~360 residues) and were quite successful: 21/23 and 18/24 variants generated with the MSA VAE and AR VAE (respectively) showing measurable functionality. The authors suggest that the lower fraction of functional variants generated by AR VAE is due to difficulty in modeling long-range dependencies. While none of the generated variants showed brighter luminescence than wild type, the authors incorporate a solubility conditioning tag to generate variants with higher solubility.

Generative Adversarial Networks: Background

Generative Adversarial Networks (GANs) are comprised of a generator network G tasked with generating examples and an adversarial discriminator D that is tasked with discriminating between generated and real examples [31]. As the generator learns to generate examples that are increasingly similar to real examples, the discriminator must also learn to distinguish between them. This equilibrium can be written as a minimax game between the Generator G and Discriminator D, where the loss function is:

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_{real}(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))] \quad (1.3)$$

where the discriminator is trained to maximize the probability $D(x)$ when x comes from a distribution of real data, and minimize the probability that the data point is real ($D(G(z))$) when the data is generated ($G(z)$). While simple in concept, and often generating samples that are less noisy than VAEs [32, 33], the Nash equilibrium between the two networks can be notoriously difficult to obtain in practice [34, 35].

Generative Adversarial Networks: Examples

Nonetheless, the authors of ProteinGAN successfully trained a GAN to generate functional malate dehydrogenases [36]. After training with nearly 17,000 unique sequences (average length: 319), 24% of 20,000 sequences generated by ProteinGAN display wild-type level activity, including a variant with 106 mutations to the closest known sequence. Interestingly, although the positional entropy of the final set of sequences closely matches that of the initial input, the generated sequences expand into new structural domains as classified by CATH, suggesting structural diversity in the generated results.

While GANs succeed in generating similar examples to the dataset, it is not immediately apparent how to condition this generation for desired properties, as there is no

compressed representation that can be searched. One approach is to separate training into two stages [37]. In the first, all available sequences are used for training, while in the second, a subset of available sequences enriched in the desired property is used to further train and bias the GAN. Amimemeur and coauthors train Wasserstein GANs [38] on 400,000 heavy or light chain sequences from human antibodies to generate 148 length regions of the respective chain. After initial training, by biasing further input data on desired properties (length, size of a negatively-charged region, isoelectronic point, and estimated immunogenicity), the estimated properties of the generated examples shifts in the desired direction. While it is difficult to get a sense for what fraction of the 100,000 generated constructs is functional from the experimental validation, extensive biophysical characterization of two of the successful designs show promising signs of retaining the designed properties *in vitro*. Notably, this approach of controlling the properties of GAN-generated sequences by biasing the input data can be implemented in a feedback loop, as shown in feedback GAN (FBGAN) for designing DNA sequences. The authors of FBGAN augment the GAN training cycle by introducing an additional component, a black box "function analyser", that checks sequences for desired properties before input to the critic, replacing older sequences that are less enriched in the desired property [39].

Conditional Generative Models: Background

An emerging class of models from language processing has developed from self-supervised learning of sequences. After masking portions of sequences, deep neural networks are tasked with generating the masked portions correctly, as conditioned on the unmasked regions. In the autoregressive generation setting, models are tasked with generating subsequent tokens based on previously generated tokens. The probability of a sequence can then be factorized as a product of conditional distributions:

$$p(\mathbf{x}) = \prod_{i=1}^{N} p(x_i|x_1, ..., x_{i-1}) \tag{1.4}$$

The main challenge is in capturing long-range dependencies. Three popular architectures, dilated convolution networks, recurrent neural networks (RNNs), and Transformer-based models, take different approaches. Dilated convolution networks include convolutions with defined gaps in the sequence in order to capture information across larger distances [40, 41]. RNNs attempt to capture positional information directly in the model state [42, 43], and an added memory layer is introduced in

Long Short-Term Memory (LSTM) networks to account for long-range interactions [44–46]. Finally, Transformer networks based on the attention mechanism, which computes a soft probability contribution over all positions in the sequence [47, 48], were also developed for language modeling to capture all possible interactions [49–51].

Conditional Generative Models: Applications

While there are many examples of using these approaches in prediction tasks [10, 23, 52–54], there are fewer examples in generation and even less with experimental validation, This likely results from the delay in physically verifying computational predictions, and we anticipate more examples of experimentally validated generation emerging soon.

Recently, Riesselman and coauthors applied autoregressive generative models to generate single domain antibodies (nanobodies) [55]. As the antibody's complementarity determining region (CDR) is difficult to align due to its high variation, an autoregressive strategy is particularly advantageous. With information from 100,000s of antibody sequences, the authors trained a residual dilated convolution network over 250,000 updates. While recurrent architectures (including an LSTM) were tested, exploding gradients were encountered, as is common in these recurrent architectures. After training, the authors generate over 37,000,000 new sequences by sampling amino acids at each new position in the sequence (ancestral sampling). Further clustering, diversity selection, and removal of motifs that may make expression more challenging (such as glycosylation sites and sulfur residues) enabled the researchers to winnow this number below 200,000, for which experimental results are pending.

For attention-based generative models, our group has applied the Transformer encoder-decoder model [49] to generating signal peptides [56]. Signal peptides are short (15-30 amino acid) sequences prepended to target protein sequences that signal the transport of the target sequence. After training with 25,000 pairs of target and signal peptide sequences, we then generated signal peptide sequences to test *in vitro*, finding that roughly half of the generated sequences secrete functional enzymes in *Bacillus subtilis*. While this work suffices as an early experimentally verified examples, there are many improvements that can be made, such as by introducing information by which to condition generation. In ProGen, an attention-based protein language model, Madani and coauthors have incorporated conditioning tokens that captures this relevant metadata, such as a protein's annotated function or organism

[57]. While this work does not (yet) have functional experimental validation, after training on 280 million sequences and their annotations from various databases, the authors show that computed Rosetta energies of the generated sequences are similar to that of natural sequences.

1.5 Optimization with Generative Models

While much of the existing work is designed to test the *validity* of generated sequences, eventually, the protein engineer expects *improved* sequences. An emerging approach to this optimization problem is to optimize with generative models [**angermuelle2020rmodel**, 58]. Instead of generating viable examples, this framework challenges models to generate *optimized* sequences by optimizing the parameters θ by which sequences are generated. Thus, the objective function is:

$$\arg\max_{\theta} \mathbb{E}_{p(x|\theta)} \left[P(S|x) \right] \tag{1.5}$$

where the search model denoted by $p(x|\theta)$ is a generative model distribution (such as a VAE) from which samples x can be generated as parameterized by θ, and $P(S|x)$ is a black box ("oracle") that maps sequences (x) to desired property values (S). This oracle can be experimental $\{x_i, y_i\}$ data or an alternate model trained on $\{x_i, y_i\}$ used to simulate ground truth data.

Design by Adaptive Sampling (DbAS)

Two difficulties arise in directly optimizing Equation 1.5: the parameter (θ) to be optimized appears in the expectation function, and the desired property set S will typically be small, which means $P(S|\theta)$ will be small and difficult to estimate. Brookes and coauthors address these issues through their Design by Adaptive Sampling (DbAS) algorithm by iteratively proposing parameters and property sets that converge on the maximal θ and desired property set S [58, 59]. During training, the DbAS algorithm iterates through cycles of sequence generation (by sampling $x \sim p(x|\theta)$ from the generative model), evaluating the samples (by the oracle), and updating θ, in close analogy to directed evolution. The authors validate this approach on synthetic ground truth data by training models (of a different type) on real biological data. They then show that generated sequences outperform traditional evolutionary methods (and the previously mentioned FBGAN) when restricted to a budget of 10,000 sequences. Recent refinements restrict $p(x|\theta)$ to avoid untrustworthy regions [59], focus the oracle as design moves between regions

of sequence space [60], or emphasize sequence diversity in generations [61].

Reinforcement Learning

An alternative approach [**angermuelle2020rmodel**] to model-based optimization has roots in reinforcement learning (RL) [62]. The RL framework is typically applied when a decision maker is asked to choose an action, a, that is available given the current state, s. From this action, the state changes through the transition function $p_a(s, s')$, with some reward, r. When a given state and action are independent of all previous states and actions (the Markov property), the system can be modeled with Markov decision processes. This requirement is satisfied by interpreting the protein sequence generation as a process where the sequence is generated from left to right. Thus, at each time step: the current state corresponds to the sequence as generated so far; the action to choosing the next amino acid; the transition function to adding this amino acid; and the reward to 0 until generation is complete, when the reward becomes the the fitness measurement. The action can then be decided by a policy network, which is trained to output a probability over all available actions based on the reward. Notably, the transition function is simple (adding an amino acid), so only the reward function needs to be approximated.

The major challenge under the RL framework is then determining the expected reward. To tackle this issue, Angermueller and coauthors use a *panel* of machine learning models, each learning a surrogate fitness function $\hat{f}_j$ based on available data from each round of experimentation. The subset of models from this panel that pass some threshold accuracy as empirically evaluated by cross validation is selected for use in estimating the reward, and the policy network is then updated based on the estimated reward. Thus, this algorithm enables a panel of models to potentially capture various aspects of the fitness landscape, but only use the models that have sufficient accuracy to update the policy network. The authors also incorporate a diversity metric by including a term in the expected reward for a sequence that counts the number of similar sequences previously explored.

The authors applied this framework to various biologically motivated synthetic datasets, including an antimicrobial peptide (8 - 75 amino acids) dataset as simulated with random forests. With eight rounds testing up to 250 sequences each, the authors show higher obtained fitness values compared to other methods, including DbAS and FBGAN. However, the authors also show that the proposed sequence diversity quickly drops and only the diversity term added to the expected reward prevents

it from converging to zero. In the future, balancing increased sequence diversity against staying within each model's trusted regions of sequence space [59, 61] will be necessary.

1.6 Conclusions and Future Directions

Machine learning has shown preliminary success in protein engineering, enabling researchers to access optimized sequences with unprecedented efficiency. These approaches allow protein engineers to efficiently sample sequence space without being limited to nature's accumulation of single amino acid mutations. As we continue to explore sequence space, expanding from the points that nature has kindly prepared, there is hope that we will find diverse solutions for myriad problems [63].

Many of machine learning's advances have been driven by data collection. For example, a large contribution to the current boom in deep learning can be traced back to ImageNet, a database of well-annotated images used for classification tasks [64]. For proteins, a well-maintained database of structures is the Protein Databank [65]. A well-organized biannual competition for protein structure prediction known as CASP (Critical Assessment of Protein Structure Prediction) [66] enabled machine learning to offer its contributions to the field [67]. A large database of protein sequences also exists [68] with reference clusters provided [15, 69]. However, these sequences are rarely coupled to *fitness* measurements and if so, are collected in diverse experimental conditions. While databases like ProtaBank [70] promise to organize data collected along with their experimental conditions, protein sequence design has yet to experience its ImageNet moment.

Fortunately, a wide variety of tools are being developed for collecting large amounts of data, including deep mutational scanning [17] and methods involving continuous evolution [71–73]. These techniques contain their own nuances and data artifacts that must be considered [74], and unifying across studies must be done carefully [75]. While these techniques currently apply to a subset of desired protein properties that are robustly measured, such as survival, fluorescence, and binding affinity, we must continue to develop experimental techniques if we hope to model and understand more complex traits such as enzymatic activity.

In the meantime, machine learning has enabled us to generate useful protein sequences on a variety of scales. In low to medium throughput settings, protein engineering guided by discriminative models enables efficient identification of improved sequences through the learned surrogate fitness function. In settings

with larger amounts of data, deep generative models have various strengths and weaknesses that may be leveraged depending on design and experimental constraints. In any case, by integrating machine learning with rounds of experimentation (as shown in the next chapters of this thesis), data-driven protein engineering promises to maximize the efforts from expensive lab work, enabling protein engineers to quickly design useful sequences.

1.7 Bibliography

References

1. Romero, P. A. & Arnold, F. H. Exploring protein fitness landscapes by directed evolution. *Nature Reviews Molecular Cell Biology* **10,** 866–876. doi:10.1038/nrm2805 (2009).

2. Arnold, F. H. Directed evolution: bringing new chemistry to life. *Angewandte Chemie International Edition* **57,** 4143–4148 (2018).

3. Huang, P.-S., Boyken, S. E. & Baker, D. The coming of age of de novo protein design. *Nature* **537,** 320–327. doi:10.1038/nature1994 (2016).

4. Garcia-Borrás, M., Houk, K. N. & Jiménez-Osés, G. Computational design of protein function. *Computational Tools for Chemical Biology* **3,** 87. doi:10.1039/9781788010139-00087 (2017).

5. Yang, K. K., Wu, Z. & Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nature Methods* **16,** 687–694. doi:10.1038/s41592-019-0496-6 (2019).

6. Mazurenko, S., Prokop, Z. & Damborsky, J. Machine Learning in Enzyme Engineering. *ACS Catalysis* (2019).

7. Volk, M. J. *et al.* Biosystems Design by Machine Learning. *ACS Synthetic Biology* (2020).

8. Ingraham, J., Garg, V., Barzilay, R. & Jaakkola, T. *Generative models for graph-based protein design* in *Advances in Neural Information Processing Systems* (2019), 15794–15805.

9. Sabban, S. & Markovsky, M. RamaNet: Computational de novo helical protein backbone design using a long short-term memory generative adversarial neural network. *F1000Research* **9,** 298 (2020).

10. Bepler, T. & Berger, B. Learning protein sequence embeddings using information from structure (2019).

11. Anand, N., Eguchi, R. R., Derry, A., Altman, R. B. & Huang, P. Protein sequence design with a learned potential. *bioRxiv* (2020).

12. Fox, R. J. *et al.* Improving catalytic function by ProSAR-driven enzyme evolution. *Nature Biotechnology* **25,** 338 (2007).

13. Liao, J. *et al.* Engineering proteinase K using machine learning and synthetic genes. *BMC Biotechnology* **7,** 16 (2007).

14. Biswas, S., Khimulya, G., Alley, E. C., Esvelt, K. M. & Church, G. M. Low-N protein engineering with data-efficient deep learning. *bioRxiv* (2020).

15. Suzek, B. E. *et al.* UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics* **31,** 926–932 (2015).

16. Xu, Y. *et al.* A Deep Dive into Machine Learning Models for Protein Engineering. *Journal of Chemical Information and Modeling* (2020).

17. Fowler, D. M. & Fields, S. Deep mutational scanning: a new style of protein science. *Nature Methods* **11,** 801. doi:10.1038/nmeth.3027 (2014).

18. Wierbowski, S. D. *et al.* A massively parallel barcoded sequencing pipeline enables generation of the first ORFeome and interactome map for rice. *Proceedings of the National Academy of Sciences USA.* doi:10.1073/pnas.1918068117 (2020).

19. Kingma, D. P. & Welling, M. Auto-encoding variational Bayes. *arXiv* (2013).

20. Rezende, D. J., Mohamed, S. & Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv* (2014).

21. Doersch, C. Tutorial on variational autoencoders. *arXiv* (2016).

22. Sinai, S., Kelsic, E., Church, G. M. & Nowak, M. A. Variational auto-encoding of protein sequences. *arXiv* (2017).

23. Riesselman, A. J., Ingraham, J. B. & Marks, D. S. Deep generative models of genetic variation capture the effects of mutations. *Nature Methods* **15,** 816–822 (2018).

24. Costello, Z. & Garcia Martin, H. How to Hallucinate Functional Proteins. *arXiv* (2019).

25. Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830 (2011).

26. Sohn, K., Lee, H. & Yan, X. *Learning structured output representation using deep conditional generative models* in *Advances in Neural Information Processing Systems* (2015), 3483–3491.

27. Greener, J. G., Moffat, L. & Jones, D. T. Design of metalloproteins and novel protein folds using variational autoencoders. *Scientific Reports* **8,** 1–12 (2018).

28. Andreini, C., Cavallaro, G., Lorenzini, S. & Rosato, A. MetalPDB: a database of metal sites in biological macromolecular structures. *Nucleic Acids Research* **41,** D312–d319 (2012).

29. Hawkins-Hooker, A. *et al.* Generating functional protein variants with variational autoencoders. *bioRxiv* (2020).

30. Semeniuta, S., Severyn, A. & Barth, E. A hybrid convolutional variational autoencoder for text generation. *arXiv* (2017).

31. Goodfellow, I. *et al. Generative adversarial nets* in *Advances in Neural Information Processing Systems* (2014), 2672–2680.

32. Theis, L., Oord, A. v. d. & Bethge, M. A note on the evaluation of generative models. *arXiv* (2015).

33. Dumoulin, V. *et al.* Adversarially learned inference. *arXiv* (2016).

34. Salimans, T. *et al. Improved techniques for training gans* in *Advances in Neural Information Processing Systems* (2016), 2234–2242.

35. Mescheder, L., Geiger, A. & Nowozin, S. Which training methods for GANs do actually converge? *arXiv* (2018).

36. Repecka, D. *et al.* Expanding functional protein sequence space using generative adversarial networks. *bioRxiv,* 789719 (2019).

37. Amimeur, T. *et al.* Designing Feature-Controlled Humanoid Antibody Discovery Libraries Using Generative Adversarial Networks. *bioRxiv* (2020).

38. Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein gan. *arXiv* (2017).

39. Gupta, A. & Zou, J. Feedback GAN for DNA optimizes protein functions. *Nature Machine Intelligence* **1,** 105–111 (2019).

40. Yu, F. & Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

41. Oord, A. v. d. *et al.* Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

42. Mikolov, T., Karafiát, M., Burget, L., Černocký, J. & Khudanpur, S. *Recurrent neural network based language model* in *Eleventh Annual Conference of the International Speech Communication Association* (2010).

43. Kalchbrenner, N. & Blunsom, P. *Recurrent continuous translation models* in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (2013), 1700–1709.

44. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Computation* **9,** 1735–1780 (1997).

45. Sutskever, I., Vinyals, O. & Le, Q. V. *Sequence to sequence learning with neural networks* in *Advances in Neural Information Processing Systems* (2014), 3104–3112.

46. Cho, K. *et al.* Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* (2014).

47. Bahdanau, D., Cho, K. & Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* (2014).

48. Luong, M.-T., Pham, H. & Manning, C. D. Effective approaches to attention-based neural machine translation. *arXiv* (2015).

49. Vaswani, A. *et al. Attention is all you need* in *Advances in Neural Information Processing Systems* (2017), 5998–6008.

50. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* (2018).

51. Radford, A. *et al.* Language models are unsupervised multitask learners. *OpenAI Blog* **1,** 9 (2019).

52. Alley, E. C., Khimulya, G., Biswas, S., AlQuraishi, M. & Church, G. M. Unified rational protein engineering with sequence-only deep representation learning. *bioRxiv,* 589333. doi:`https://doi.org/10.1101/589333` (2019).

53. Rives, A. *et al.* Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv,* 622803 (2019).

54. Rao, R. *et al. Evaluating protein transfer learning with TAPE* in *Advances in Neural Information Processing Systems* (2019), 9686–9698.

55. Riesselman, A. J. *et al.* Accelerating Protein Design Using Autoregressive Generative Models. *bioRxiv,* 757252 (2019).

56. Wu, Z. *et al.* Signal Peptides Generated by Attention-Based Neural Networks. *ACS Synthetic Biology* **9,** 2154–2161 (2020).

57. Madani, A. *et al.* ProGen: Language Modeling for Protein Generation. *arXiv* (2020).

58. Brookes, D. H. & Listgarten, J. Design by adaptive sampling. *arXiv* (2018).

59. Brookes, D. H., Park, H. & Listgarten, J. Conditioning by adaptive sampling for robust design. *arXiv* (2019).

60. Fannjiang, C. & Listgarten, J. Autofocused oracles for model-based design. *arXiv preprint arXiv:2006.08052* (2020).

61. Linder, J., Bogard, N., Rosenberg, A. B. & Seelig, G. A Generative Neural Network for Maximizing Fitness and Diversity of Synthetic DNA and Protein Sequences. *Cell Systems* **11,** 49–62 (2020).

62. Sutton, R. S. & Barto, A. G. *Reinforcement learning: An introduction* (MIT press, 2018).

63. Nobeli, I., Favia, A. D. & Thornton, J. M. Protein promiscuity and its implications for biotechnology. *Nature Biotechnology* **27,** 157–167 (2009).

64. Deng, J. *et al. Imagenet: A large-scale hierarchical image database* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2009), 248–255.

65. Berman, H. M. *et al.* The protein data bank. *Nucleic Acids Research* **28,** 235–242 (2000).

66. Moult, J., Pedersen, J. T., Judson, R. & Fidelis, K. A large-scale experiment to assess protein structure prediction methods. *Proteins: Structure, Function, and Bioinformatics* **23,** ii–iv (1995).

67. Senior, A. W. *et al.* Improved protein structure prediction using potentials from deep learning. *Nature,* 1–5 (2020).

68. Consortium, U. UniProt: a worldwide hub of protein knowledge. *Nucleic Acids Research* **47,** D506–d515 (2019).

69. Suzek, B. E., Huang, H., McGarvey, P., Mazumder, R. & Wu, C. H. UniRef: comprehensive and non-redundant UniProt reference clusters. *Bioinformatics* **23,** 1282–1288 (2007).

70. Wang, C. Y. *et al.* ProtaBank: A repository for protein design and engineering data. *Protein Science* **27,** 1113–1124. doi:10.1002/pro.3406 (2018).

71. Esvelt, K. M., Carlson, J. C. & Liu, D. R. A system for the continuous directed evolution of biomolecules. *Nature* **472,** 499–503 (2011).

72. Morrison, M. S., Podracky, C. J. & Liu, D. R. The developing toolkit of continuous directed evolution. *Nature Chemical Biology* **16,** 610–619 (2020).

73. Zhong, Z. *et al.* Automated continuous evolution of proteins in vivo. *ACS Synthetic Biology* (2020).

74. Eid, F.-E. *et al.* Systematic auditing is essential to debiasing machine learning in biology. *bioRxiv* (2020).

75. Dunham, A. & Beltrao, P. Exploring amino acid functions in a deep mutational landscape. *bioRxiv*. doi:10.1101/2020.05.26.116756 (2020).

Chapter 2

MACHINE LEARNING-GUIDED COMBINATORIAL MUTAGENESIS FOR EVOLVING STEREODIVERGENCE

1. Wu, Z., Kan, S. B. J., Lewis, R. D., Wittmann, B. J. & Arnold, F. H. Machine learning-assisted directed protein evolution with combinatorial libraries. *Proceedings of the National Academy of Sciences USA* (2019).

Contributions Statement: Z.W. performed all experiments and data analyses.

2.1 Abstract

To reduce experimental effort associated with directed protein evolution and to explore the sequence space encoded by mutating multiple positions simultaneously, we incorporate machine learning into the directed evolution workflow. Combinatorial sequence space can be quite expensive to sample experimentally, but machine-learning models trained on tested variants provide a fast method for testing sequence space computationally. We provide an example application in evolving an enzyme to produce each of the two possible product enantiomers (i.e., stereodivergence) of a new-to-nature carbene Si–H insertion reaction. The approach predicted libraries enriched in functional enzymes and fixed seven mutations in two rounds of evolution to identify variants for selective catalysis with 93% and 79% *ee* (enantiomeric excess). By greatly increasing throughput with *in silico* modeling, machine learning enhances the quality and diversity of sequence solutions for a protein engineering problem.

2.2 Introduction

Nature provides countless proteins with untapped potential for technological applications. Rarely optimal for their envisioned human uses, nature's proteins benefit from sequence engineering to enhance performance. Successful engineering is no small feat, however, as protein function is determined by a highly tuned and dynamic ensemble of states [1]. In some cases, engineering to enhance desirable features can be accomplished reliably by directed evolution, in which beneficial mutations are identified and accumulated through an iterative process of mutation and testing hundreds to thousands of variants in each generation [2–4]. However, implementing a suitable screen or selection can represent a significant experimental burden.

Given that screening is the bottleneck and most resource intensive step for the majority of directed evolution efforts, devising ways to screen protein variants *in silico* is highly attractive. Molecular dynamics simulations, which predict dynamic structural changes for protein variants, have been used to predict changes in structure [5] and protein properties caused by mutations [6]. However, full simulations are also resource intensive, requiring hundreds of CPU hours for each variant, a mechanistic understanding of the reaction at hand, and ideally, a reference protein structure. A number of other, less computationally intensive physical models have also been used to identify sequences likely to retain fold and function for further experimental screening [7–9].

An emerging alternative for screening protein function *in silico* is machine learning, which comprises a set of algorithms that make decisions based on data [10]. By building models directly from data, machine learning has proven to be a powerful, efficient, and versatile tool for a variety of applications, such as extracting abstract concepts from text and images or beating humans at our most complex games [11, 12]. Previous applications of machine learning in protein engineering have identified beneficial mutations [13] and optimal combinations of protein fragments [14] for increased enzyme activity and protein stability, as reviewed recently [15]. Here we use machine learning to enhance directed evolution, using combinatorial libraries of mutations to explore sequence space more efficiently than conventional directed evolution with single mutation walks. The size of a mutant library grows exponentially with the number of residues considered for mutation and quickly becomes intractable for experimental screening. However, by leveraging *in silico* models built based on sampling of a combinatorial library, machine learning assists directed evolution to make multiple mutations simultaneously and traverse fitness landscapes more efficiently.

In the machine learning-assisted directed evolution strategy presented here, multiple amino acid residues are randomized in each generation. Sequence-function information sampled from the large combinatorial library is then used to predict a restricted library with an increased probability of containing variants with high fitness. The best-performing variants from the predicted libraries are chosen as the starting points for the next round of evolution, from which further improved variants are identified. Here, we use machine learning-assisted directed evolution to engineer an enzyme for stereodivergent carbon–silicon bond formation, a new-to-nature chemical transformation.

2.3 Results

Machine Learning in Directed Evolution

In directed evolution, a library of variants is constructed from parental sequences, screened for desired properties, and the best variants are used to parent the next round of evolution; all other variants are discarded. When machine learning assists directed evolution, sequences and screening data from all the variants can be used to train a panel of models (covering linear, kernel, neural network, and ensemble methods (SI Appendix, Model Training)). The models with highest accuracy are then used to screen variants in a round of *in silico* evolution, where the models simulate the fitness values of all possible sequences and rank the sequences by fitness. A restricted library containing the variants with the highest predicted fitness values is then constructed and screened experimentally.

This work explores the full combinatorial space of mutations at multiple positions. Figure 2.1 illustrates the approach considering a set of four mutated positions. In a conventional directed evolution experiment with sequential single mutations, identifying optimal amino acids for N positions in a set requires N rounds of evolution (Figure 2.1A). Machine learning-assisted evolution samples the combinatorial space of co-mutated poitions *in silico*, enabling larger steps through sequence space in each round (Figure 2.1B). In this approach, data from a random sample of the combinatorial library, the input library, are used to train machine learning models. These models are used to predict a smaller set of variants, the predicted library, which can be encoded with degenerate codons to test experimentally [16]. The best-performing variant is then used as the parent sequence for the next round of evolution with mutations at new positions.

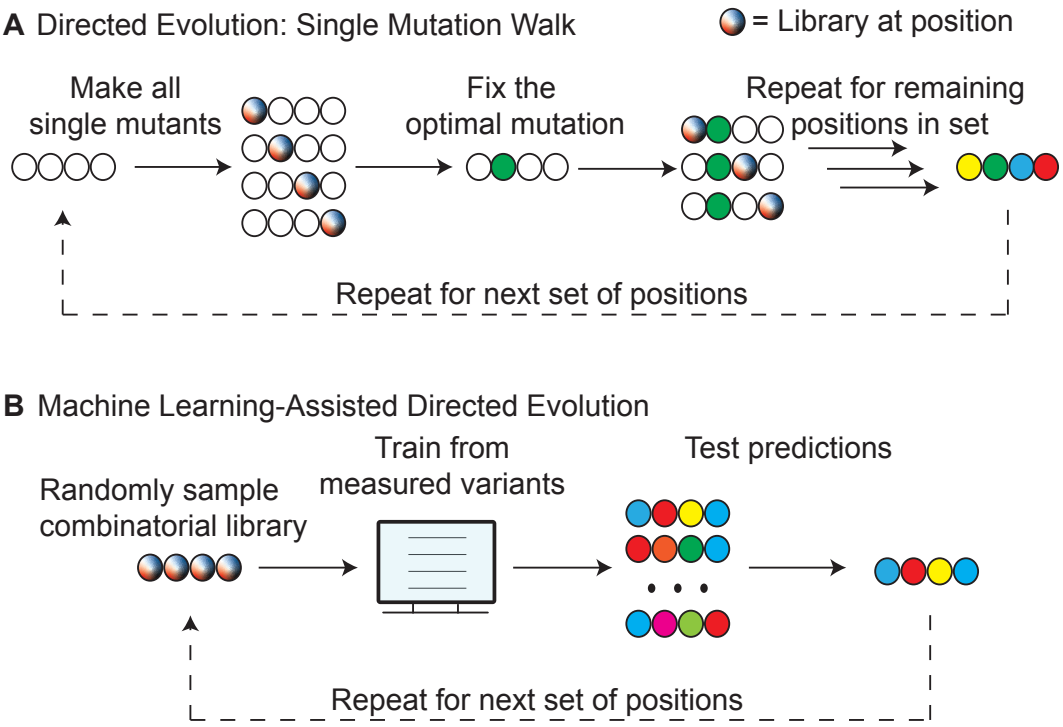


Figure 2.1: (A) Directed evolution with single mutations. If limited to single mutations, identifying optimal amino acids for N positions requires N rounds of evolution. (B) Machine learning-assisted directed evolution. Due to increased throughput provided by screening *in silico*, four positions can be explored simultaneously in a single round, enabling a broader search of sequence-function relationships and deeper exploration of epistatic interactions.

Application to Evolution of Enantiodivergent Enzyme Activity

We next used machine learning-assisted directed evolution to engineer an enzyme to produce each of two possible product enantiomers. For the purpose of this demonstration, we selected the reaction of phenyldimethyl silane with ethyl 2-diazopropanoate (Me-EDA) catalyzed by a putative nitric oxide dioxygenase from *Rhodothermus marinus* (*Rma* NOD), as shown in Figure 2.2. Carbon–silicon bond formation is a new-to-nature enzyme reaction [17], and *Rma* NOD with mutations Y32K and V97L catalyzes this reaction with 76% *ee* for the (*S*)-enantiomer in whole-cell reactions (Table 2.4.

Silicon has potential for tuning the pharmaceutical properties of bioactive molecules [18, 19]. Because enantiomers of bioactive molecules can have stark differences in their biological effects [20], access to both is important [21]. Screening for enantioselectivity, however, typically requires long chiral separations to discover beneficial mutations in a low-throughput screen [22]. We thus tested whether machine

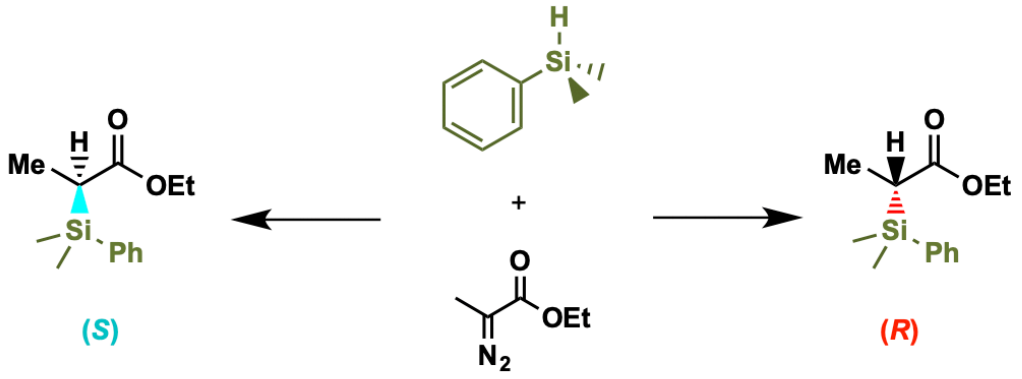


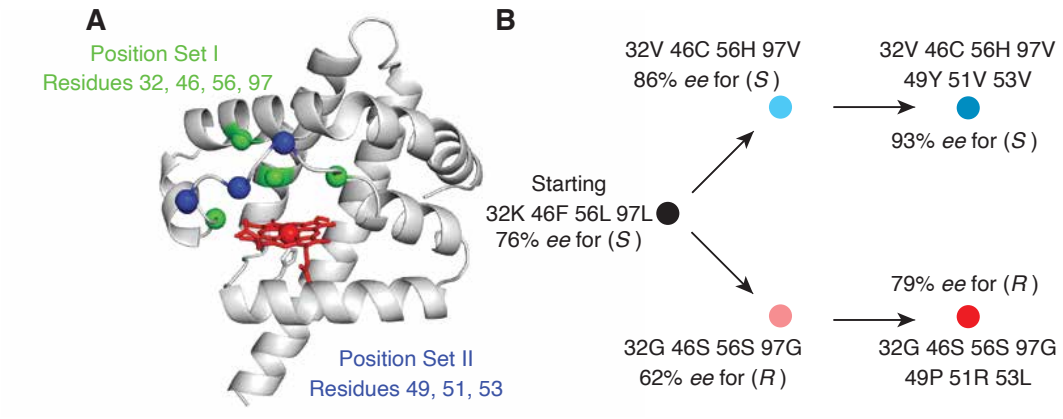
Figure 2.2: Carbon–silicon bond formation catalyzed by heme-containing nitric oxide dioxygenase from *R. marinus* to form individual product enantiomers with high selectivity.

learning-assisted directed evolution can efficiently generate two catalysts to make each of the product (*S*)- and (*R*)-enantiomers starting from a single parent sequence.

We chose the parent *Rma* NOD (UniProt ID: D0MGT2) [23] enzyme for two reasons. First, *Rma* NOD is native to a hyperthermophile and should be thermostable. Because machine learning-assisted directed evolution makes multiple mutations per iteration, a starting sequence capable of accommodating multiple potentially destabilizing mutations is ideal [24]. Second, while we previously engineered a cytochrome *c* (*Rma* cyt *c*) to >99% *ee* for the (*R*)-enantiomer, wild-type *Rma* cyt c serendipitously started with 97% *ee* [17]. We hypothesized that a parent enzyme with less enantiopreference (76% *ee* for the (*S*)-enantiomer in whole cells) would be a better starting point for engineering enantiodivergent variants.

During evolution for enantioselectivity, we sampled two sets of amino acid positions: Set I contained mutations to residues K32, F46, L56, and L97, and Set II contained mutations to residues P49, R51, and I53 after fixing beneficial mutations identified from Set I. For both sets, we first tested and sequenced an initial set of randomly selected mutants (the input library) to train models. We next tested a restricted set of mutants predicted to have high selectivity (the predicted library). The targeted positions are shown in a structural homology model in Figure 2.3A. Set I positions were selected based on proximity to the putative active site, while Set II positions were selected based on their proximity to the putative substrate entry channel.

Machine learning models are more useful when trained with data broadly distributed across input space, even if those data are noisy [26]. When designing a training set



C

	Variants Input	Fraction of Total Library	Leave-One-Out CV Score	Predictions Tested
Position Set I	124	0.60 %	0.549	90 for (R) 90 for (S)
Position Set II from VCHV	155	8.3 %	0.630	90
Position Set II from GSSG	166	8.9 %	0.605	90

Figure 2.3: (A) Structural homology model of *Rma* NOD and positions of mutated residues made by SWISS-MODEL [25]. Set I positions 32, 46, 56, and 97 are shown in red, and Set II positions 49, 51, and 53 are shown in blue. (B) Evolutionary lineage of the two rounds of evolution. (C) Summary statistics for each round, including the number of sequences obtained to train each model, the fraction of the total library represented in the input variants, each model's Leave-One-Out Pearson correlation, and the number of predicted sequences tested.

for machine learning-assisted directed evolution, it is thus important to maximize the input sequence diversity by avoiding disproportionate amino acid representation (e.g. from codon usage). We therefore used NDT codons for the input libraries. NDT libraries encode 12 amino acids having diverse properties with 12 unique codons [22], thus minimizing the probability that an amino acid is overrepresented in the initial training set [27]. Notably, the parent amino acid at a site is still considered by the model even if it is not encoded by the NDT codons, as sequence-function data are available for the parent sequence.

The evolution experiment is summarized in Figure 2.3B. In the first round, *Rma* NOD Y32K V97L (76% *ee*) was used as a parent for NDT mutagenesis at the Set I positions. From 124 sequence–function relationships sampled randomly, models were trained to predict a restricted set of selective variants. Specifically, a variety of models covering linear, kernel, shallow neural network, and ensemble methods were tested on each library, from which the optimum models were used to rank every

sequence in the theoretical library by its predicted fitness. Under strict limitations in experimental throughput, and with one 96-well plate as the smallest batch size, we settled on two plates of input data for each round of evolution, and one plate of tested predictions. However, increased throughput allows for increased likelihood of reaching the landscape's optimum as discussed in Chapter 3. The lower numbers of variants input in Figure 2.3 compared to two full 96-well plates of sequencing reflect failed DNA sequencing reads of these two plates.

From the predicted libraries for both enantiomers, two variants, called VCHV (93% *ee*) and GSSG (62% *ee*) for their amino acids at positions 32, 46, 56, and 97, were identified by screening 90 variants for each. Variants VCHV and GSSG were then used as the parent sequences for the second round of mutation at the three positions in Set II. VCHV was the most selective variant in the initial screening, but it was less selective in final validation. The approach of experimentally testing a library predicted by models trained on a randomly sampled input library was repeated. From those predicted libraries, we obtained two variants with measured enantioselectivities of 93% and 79% *ee* for the (*S*)- and (*R*)-enantiomers, respectively. These two enantioselective enzymes were achieved after obtaining 445 sequence-function relationships for model training and testing an additional 360 predicted variants, for a total of 805 variants tested experimentally covering seven positions, as summarized in Figure 2.3C.

Machine Learning Identifies Diverse Improved Sequences

While a comparison on an empirical landscape in Chapter 3 shows that machine learning-assisted directed evolution is more likely than directed evolution alone to identify improved variants, yet another benefit of this approach is the ability to identify a diverse set of sequences for accomplishing a specific task. Having diverse solutions is attractive as some of those variants may satisfy other design requirements, such as increased total activity, altered substrate tolerance, specific amino acid handles for further protein modification, or sequence diversity for intellectual property considerations [28]. By enabling exploration of the combinatorial space, machine learning-assisted directed evolution is able to identify multiple solutions for each engineering objective.

Table 2.1 and Table 2.2 summarize the most selective variants in the input and predicted libraries for position Sets I and II. The input library for Set I is the same for both product enantiomers. The parent sequences for Set II, VCHV and GSSG,

are highlighted in cyan and red, respectively, in the tables. The improvement in total activity measured in whole *E. coli* cells compared to the starting variant (32K, 46F, 56L, 97L) obtained after two rounds of machine learning-assisted directed evolution is also shown in Table 2.2. Although evolved for enantioselectivity, the variants have increased levels of activity in whole-cells. Negative controls with cells expressing non-heme proteins yield a racemic mixture of product enantiomers, due to a low level of nonselective background activity from free heme or heme proteins. For the screen to report higher selectivity, the protein variant must not only exhibit higher selectivity, but do so with enough activity to overcome the background racemic activity. Thus, enhanced activity in the presence of mild selectivity is one path to higher selectivity. The two variants most selective for the (*S*)-enantiomer, 49P 51V 53I and 49Y 51V 53V from VCHV, differ by less than 1% in ee. However, the 49P 51V 53I variant has 14% higher total activity under screening conditions. By providing multiple solutions in a combinatorial space for a single design criterion, machine learning is able to identify variants with other beneficial properties.

The solutions identified by this approach can also be non-obvious. For example, the three most (*S*)-selective variants in the initial input for Position Set I are YNL**L**, CSV**L**, and CVH**V**. The three most selective sequences from the restricted, predicted library are VGV**L**, CFN**L**, and VCH**V**. If only considering the last residue in bold, the predicted library can be sampled from the top variants in the input library. However, for each of the other three positions, there is at least one mutation that is not present in the top three input sequences.

Table 2.1: Summary of the most (*S*)- and (*R*)-selective variants in the input and predicted libraries in Set I (K32, F46, L56, L97). The parent sequences used for Set II for (*S*)- and (*R*)-selectivity are shown in cyan and red, respectively.

Set I: Residues 32, 46, 56, and 97

Input Variants					Predicted Variants				
Residue				**Selectivity**	**Residue**				**Selectivity**
32	46	56	97	% *ee*	32	46	56	97	% *ee*
Y	N	L	L	84 % (*S*)	V	G	V	L	90 % (*S*)
C	S	V	L	83 % (*S*)	C	F	N	L	90 % (*S*)
C	V	H	V	82 % (*S*)	V	C	H	V	86 % (*S*)
C	R	S	G	56 % (*R*)	G	S	S	G	62 % (*R*)
I	S	C	G	55 % (*R*)	G	F	L	R	24 % (*R*)
N	V	R	I	47 % (*R*)	H	C	S	R	17 % (*R*)

Table 2.2: Summary of the most (*S*)- and (*R*)-selective variants in the input and predicted libraries in Position Set II (P49, R51, I53). Mutations that improve selectivity for the (*S*)-enantiomer appear in the background of [32V, 46C, 56H, 97V (VCHV)] and for the (*R*)-enantiomer are in [32G, 46S, 56S, 97G (GSSG)]. Activity increase over the starting variant, 32K, 46F, 56L, 97L (KFLL), is shown for the final variants. The parent sequences used for evolving for (*S*)- and (*R*)-selectivity are highlighted in cyan and red, respectively.

	Set II: Residues 49, 51, 53								
	Input Variants				Predicted Variants				
	Residue			**Selectivity**	**Residue**			**Selectivity**	Cellular activity
	49	51	53	% *ee*	49	51	53	% *ee*	increase over KFLL
Evolved	P	R	I	86 % (*S*)	Y	V	V	93 % (*S*)	2.8-fold
from	Y	V	F	86 % (*S*)	P	V	I	93 % (*S*)	3.2-fold
VCHV	N	D	V	75 % (*S*)	P	V	V	92 % (*S*)	3.1-fold
Evolved	P	R	I	62 % (*R*)	P	R	L	79 % (*R*)	2.2-fold
from	Y	F	F	57 % (*R*)	P	G	L	75 % (*R*)	2.1-fold
GSSG	C	V	N	52 % (*R*)	P	F	F	70 % (*R*)	2.2-fold

Predicted Regions of Sequence Space are Enriched in Function

While directed evolution assisted with machine learning-assisted is more likely to reach sequences with higher fitness than without, as demonstrated in Chapter 3, there may well be instances where other evolution strategies serendipitously discover variants with higher fitness more quickly. Therefore, since the purpose of library creation is to increase likelihood of finding an improved variant, we caution against focusing solely on examples of individual variants with higher fitness and propose an alternative analysis.

Sequence-fitness landscapes are typically represented with fitness values on the vertical axis, dependent on some ordering of the corresponding protein sequences. Representing this high-dimensional space, even when it is explored with single mutations, is complicated and requires sequencing each variant [29]. However, in functional protein space, the engineer is primarily concerned with fitness. Therefore, an alternative representation of a library is a 1-dimensional distribution of fitness values sampled at random for each encoded library. In other words, the sequences are disregarded for visualization, and the library is represented by the distribution of its fitness values. This is the critical paradigm shift presented by this visualization. While protein engineers design in sequence space, the ultimate goal is to achieve higher fitness. To this end, a library can be visually represented by plotting only their fitness values, which can be well approximated by obtaining random samples from

the library.

We apply this visualization to each subplot in Figure 2.4, which plots experimentally obtained measurements for both the input libraries (lighter color) and the predicted libraries (darker color). In addition to the individual selectivity values plotted as a rug plot along the abscissa, we provide Gaussian kernel density estimates as visualizations of the fitness distribution in each library. In the ideal case for each subplot, the darker predicted regions should be shifted toward higher selectivity, as compared to the lighter regions describing the input libraries. This representation is able to show the main benefit of incorporating machine learning into directed evolution, which is the ability to focus expensive experiments on regions of sequence space enriched in desired variants.

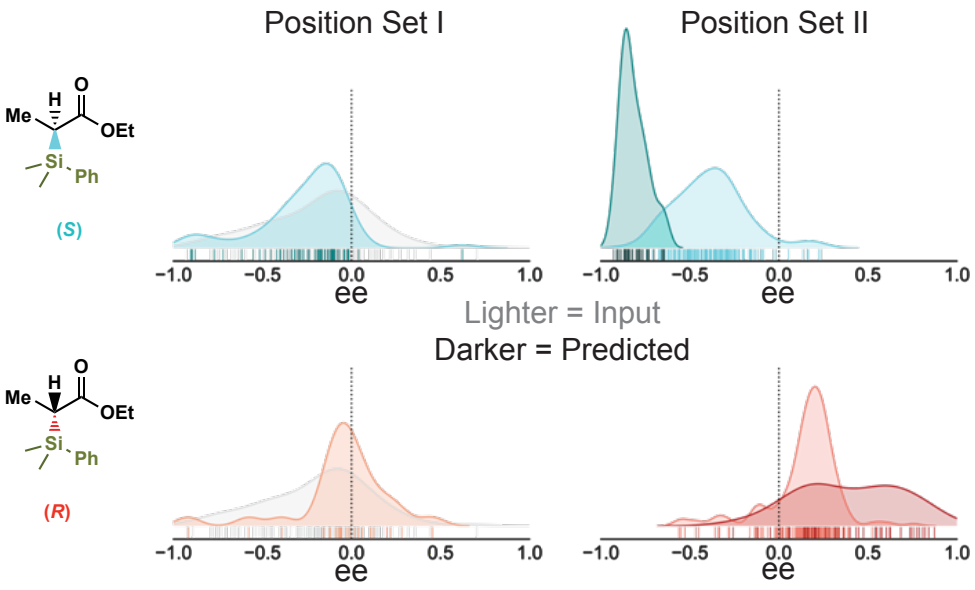


Figure 2.4: A library's fitness values can be visualized as a 1-dimensional distribution, in this case as kernel density estimates over corresponding rug plots. This figure shows subplots for each library illustrating the changes between input (lighter) and predicted (darker) libraries for the (*S*)- (cyan) and (*R*)-enantiomers (red). The initial input library for Set I is shown in gray. The predicted (darker) libraries for each round are shifted toward the right and left of the distributions for the (*S*)- and (*R*)-enantiomers, respectively. For reference, dotted lines are shown for no enantiopreference (0 % *ee*).

In applying this visualization to our evolution strategy, a few conclusions may be drawn. First, the distribution of random mutations made in the input libraries is

shifted toward (*R*)- and (*S*)-selectivity depending on the starting variant, as has been shown previously [30]. In other words, random mutations made from an (*R*)-selective variant are more likely to be (*R*)-selective. More importantly, the machine-learning algorithm is able to focus its predictions on areas of sequence space that are enriched in high fitness, as can be seen in the shift in distribution from the lighter input libraries to the darker predicted libraries toward higher selectivity values. Notably, the machine learning algorithms appear to have more pronounced benefits in Position Set II, likely due to the smaller number of positions explored and larger number of sequence-function relationships obtained. Nonetheless, machine learning optimized directed evolution by sampling regions of sequence space dense in functionality.

2.4 Discussion

We have shown that machine learning can be used to quickly screen a full recombination library *in silico* using sequence-fitness relationships randomly sampled from the library. The predictions for the most-fit sequences are useful when incorporated into directed evolution. By sampling large regions of sequence space *in silico* to reduce *in vitro* screening efforts, we rapidly evolved a single parent enzyme to generate variants that selectively form both product enantiomers of a new-to-nature C–Si bond-forming reaction. Rather than relying on identifying beneficial single mutations as other methods such as ProSAR do [13], we modeled epistatic interactions at the mutated positions by sampling the combinatorial sequence space directly and incorporating models with nonlinear interactions.

Machine learning increases effective throughput by providing an efficient computational method for estimating desired properties of all possible proteins in a large library. Thus we can take larger steps through sequence space by identifying combinations of beneficial mutations, circumventing the need for indirect paths [31] or alterations of the nature of selection [29], and potentially avoiding negative epistatic effects resulting from the accumulation of large numbers of mutations [32] that require reversion later in the evolution [33]. This gives rise to novel protein sequences that would not be found just by recombining the best amino acids at each position. Allowing simultaneous incorporation of multiple mutations accelerates directed evolution by navigating different regions of the fitness landscape concurrently and avoiding scenarios where the search for beneficial mutations ends in low-fitness regions of sequence space.

Importantly, machine learning-assisted directed evolution also results in solutions

that appear quite distinct. For example, proline is conserved at residue 49 in two of the most (*S*)-selective variants. Proline is considered unique for the conformational rigidity it confers and at first may seem structurally important, if not critical for protein function. However, tyrosine and arginine are also tolerated at position 49 with less than 1% loss in enantioselectivity. This suggests that there are diverse solutions in protein space for specific properties, as has also recently been shown in protein design [8]. Computational models make abstractions to efficiently model physical processes, and the level of abstraction must be tailored to the task, such as protein structure prediction [34]. While predictive accuracy could be improved by more computationally expensive simulations or by collecting more data for machine learning, improved variants can already be identified by sampling from a space predicted to be dense in higher fitness variants. Nevertheless, full datasets collected with higher throughput methods such as deep mutational scanning [35] serve as valuable test beds for validating the latest machine-learning algorithms for both regression [36, 37] and design [38] that require more data.

An evolution strategy similar in spirit to that described here was recently applied to the evolution of GFP fluorescence [39]. However, the implementations are quite different. Saito and coworkers used Gaussian processes to rank sequences based on their probability of improvement, or the probability that a variant outperforms those in the training set .We take a different approach of identifying the optimal variants, focusing efforts in the area of sequence space with highest fitness. Additionally, because it is difficult to know *a priori* which models will be most accurate for describing a particular landscape, we tested multiple types of models, from linear to ensemble models, to predict the optimal sequences. Modeling the effects of previously identified single amino acid mutations has also recently been studied for evolution of enantioselectivity of an enzyme [40]. This study and others focused on increasing the accuracy of protein modeling by developing other physical descriptors [41, 42] or embedded representations [43] suggest that machine learning will assist directed evolution beyond the baseline implementation employed here.

By providing an efficient estimate for desired properties, machine learning models are able to leverage the information from limited experimental resources to model proteins, without the need for a detailed understanding of how they function. Machine learning-assisted directed evolution with combinatorial libraries provides a tool for understanding the protein sequence-function relationship and for rapidly engineering useful proteins. Protein engineers have been sentenced to long treks through sequence

space in the search for improved fitness. Machine learning can help guide us to the highest peaks.

2.5 Materials and Methods

Library Cloning, Expression, and Characterization of *Rma* NOD

The gene encoding *Rma* NOD was obtained as a gBlock and cloned into pET22b(+) (Novagen catalog number 69744). Standard PCR amplification and Gibson assembly were used for libraries with degenerate codons specified by SwiftLib [16]. Encoded versus sequenced codon distributions are shown in Figure 2.8. Expression was performed in 96-well deep-well plates in 1-mL HyperBroth (AthenaES) using *Escherichia coli* BL21 E. cloni EXPRESS (Lucigen) with 100 µL/mL ampicillin from a 20-fold dilution of overnight culture. Expression cultures were induced after 2.5 h of outgrowth with 0.5 mM IPTG (Isopropyl β-D-1-thiogalactopyranoside), and heme production was enhanced with supplementation of 1 mM 5-aminolevulinic acid.

The relative product activity was measured using 10 mM Me-EDA and 10 mM PhMe2SiH with whole *E. coli* cells resuspended in 400 µl nitrogen-free M9-N buffer, pH 7.4 (47.7 mM Na_2HPO_4, 22.0 mM KH_2PO_4, 8.6 mM NaCl, 2.0 mM $MgSO_4$, and 0.1 mM $CaCl_2$). Reactions were incubated anaerobically at room temperatures for 6 hr, before extraction into 600 µl cyclohexane. Enantiomeric excess was measured by running the organic solution on a JACSO 2000 series supercritical fluid chromatography (SFC) system with a Chiralcel OD-H (4.6 mm x 25 cm) chiral column (95% CO_2, 5% isopropanol, 3 minutes).

Rma NOD Model Training and Prediction Testing

Screening information was paired with protein sequence data obtained from rolling circle amplification followed by sequencing by MCLab. The sequence-function pairs, available on ProtaBank [44], were used to train a panel of models with default hyperparameters in the scikit-learn Python package [45], including K-nearest neighbors, linear (including Automatic Relevance Detection, Bayesian Ridge, Elastic Net, Lasso LARS, and Ridge), decision trees, random forests (including AdaBoost, Bagging, and Gradient Boosting), and multilayer perceptrons. The top 3 model types were selected, and gridsearch cross-validation was used to identify the optimal hyperparameters. The top 3 hyperparameter sets for the top 3 model types were used to identify the top 1000 sequences in each predicted library. Degenerate codons encoding amino acids occurring with highest frequencies in every model at each

position were identified by Swiftlib [16], and 90 random variants were tested *in vitro*. This random sampling differs from that in the empirical fitness landscape, where all sequences have been enumerated and can be easily tested. While sampling randomly means we may not have tested the optimal sequence as identified in trained models, we are able to generate fitness distributions as in Figure 2.4 to describe this space.

(A) Plasmid construction

All variants described in this study were cloned and expressed using the pET22(b)+ vector (MilliporeSigma, St. Louis, MO). The gene encoding wild-type *Rhodothermus marinus* putative nitric oxide dioxygenase (*Rma* NOD, UniProt ID [23]: D0MGT2 _RHOM4) was obtained as a single gBlock (Integrated DNA Technologies, Coralville, IA), codon-optimized, and cloned using Gibson assembly [46] into pET22(b)+ with a 6xHisTag appended at the C-terminus. This plasmid was transformed into *E. cloni* EXPRESS BL21(DE3) cells (Lucigen, Middleton, WI).

DNA coding sequence of *Rma* NOD 32K 97L with a C-terminal 6xHisTag:

ATGGCGCCGACCCTGTCGGAACAGACCCGTCAGTTGGTACGTGCG
TCTGTGCCTGCACTGCAGAAACACTCAGTCGCTATTAGCGCCACG
ATGTATCGGCTGCTTTTCGAACGGTATCCCGAAACGCGGAGCTTAT
TTGAACTTCCTGAGAGACAGATACACAAGCTTGCGTCGGCCCTGT
TGGCCTACGCCCGTAGTATCGACAACCCATCGGCGTTACAGGCGG
CCATCCGCCGCATGGTGCTTTCCCACGCACGCGCAGGAGTGCAGG
CCGTCCATTATCCGCTGGTTTGGGAATGTTTGAGAGACGCTATAA
AAGAAGTCCTGGGCCCGGATGCCACCGAGACCCTTCTGCAGGCGT
GGAAGGAAGCCTATGATTTTTTAGCTCATTTACTGTCTACCAAGGA
AGCGCAAGTCTACGCTGTGTTAGCTGAACTCGAGCACCACCACCA
CCACCACTGA

Amino acid sequence of *Rma* NOD 32K 97L with a C-terminal 6xHisTag

MAPTLSEQTRQLVRASVPALQKHSVAISATMYRLLFERYPETRSLFEL
PERQIHKLASALLAYARSIDNPSALQAAIRRMVLSHARAGVQAVHYP
LVWECLRDAIKEVLGPDATETLLQAWKEAYDFLAHLLSTKEAQVYA
VLAELEHHHHHH

(B) Protein expression

Single colonies from Luria Broth (LB)-ampicillin (100 μg/mL) agar plates were picked using sterile toothpicks and grown in 600 μL LB-ampicillin in 2-mL 96-well deep-well plates at 37 °C, 250 rpm, 80% humidity overnight (12—18 hours). Multi-channel pipettes were used to transfer 50 μL of overnight culture into four deep-well plates containing 1 mL Hyperbroth (HB, AthenaES) each. Four replicates of each well position were made to minimize variability in cell culture and maximize accuracy for downstream modeling. The expression plate were incubated at 37 °C, 250 rpm, 80% humidity for 2.5 h. The plates were then chilled on ice for 30 minutes and induced with 0.5 mM isopropyl β-D-1-thiogalactopyranoside and supplemented with 1 mM 5-aminolevulinic acid to increase heme production. The plate was incubated at 22°C and 220 rpm overnight. The plate was then centrifuged at 3000g for 10 minutes at 4 °C. Each individual well was resuspended in 100 μL M9-N buffer (pH 7.4, 47.7 mM Na_2HPO_4, 22.0 mM KH_2PO_4, 8.6 mM NaCl, 2.0 mM $MgSO_4$, and 0.1 mM $CaCl_2$). The four replicates were combined for 400 μL total in M9-N buffer.

(C) Biocatalytic reaction and assay

In an anaerobic chamber, 10 μL of 400 mM $PhMe_2SiH$ (in acetonitrile) and 10 μL of 400 mM ethyl-2-diazopropanoate (Me-EDA, in acetonitrile) were added to 380 μL whole cells resuspended in M9-N buffer. The final concentrations in each well were 10 mM Me-EDA and 10 mM $PhMe_2SiH$ in each 400 μL reaction mix. The reaction plate was covered with a foil cover (USA scientific) and shaken at 1000 rpm for 4 h. Six hundred μL of cyclohexane were added with a multi-channel pipette to each well to quench the reaction and extract the reaction products, which have been previously characterized [17]. Plates were centrifuged to remove cells (3000 g, 10 minutes) and enantiomeric excess was measured by running the organic solution on a JACSO 2000 series supercritical fluid chromatography (SFC) system with a Chiralcel OD-H (4.6 mm x 25 cm) chiral column (95% CO_2, 5% isopropanol, 3 minutes). Final variants in Table 2.1 and Table 2.2 were expressed and tested in biological triplicate (in addition to the previous protocol of combining four replicates). Automatic integration was performed in ChemStation.

(D) Model training

Machine-learning models were trained with sequencing information from MCLAB Inc and enantiomeric data obtained by SFC. To model the data, the following regressors from the superlative scikit-learn package [45] were used: K-nearest neighbors, linear (including Automatic Relevance Detection, Bayesian Ridge, Elastic Net, Lasso LARS, and Ridge), decision trees, random forests (including AdaBoost, Bagging, and Gradient Boosting), and multilayer perceptrons, as it is difficult to know *a priori* which model will best fit the landscape. For example, if the selected positions are truly non-interactive, we can expect much of the landscape's variance to be explained by a linear model. However, for more epistatic landscapes, we must account for this nonlinearity. Therefore, many different model classes were tested, all of which can be run (with hyperparameter optimization) on a personal MacBook Pro in less than one day. The three model types with highest Pearson correlation from a Leave-One-Out cross validation (LOO CV) with default hyperparameters were selected for gridsearch hyperparameter optimization. From this gridsearch, the three sets of hyperparameters with highest LOO CV Pearson correlation were selected, for a total of nine models in order to capture different characteristics of the landscape with relatively low accuracy models. The models were retrained on the full dataset and used for predicting a restricted library, discussed in the section below.

(E) Model predictions

Directly synthesizing the DNA encoding the top variants is quite expensive. Therefore, we interpret our models' predictions by the frequency of each amino acid's occurrence in a top fraction (the top 1000) of the library, which we are able to encode efficiently with degenerate codon libraries. An example is shown in Table 2.3. For cloning purposes, at this point the sequence information predicted from the models is lost, as each position is considered independently to reduce DNA synthesis and subcloning costs. Additionally, we elected to include all 20 amino acids in the predictions even though less than 20 were encoded in the input libraries, to provide an estimate for when the models may be predicting high fitness based on mutations at other positions. A full description of this step is provided with the accompanying Table 2.3.

(F) Experimental validation of predictions

The top amino acids at each position are encoded by degenerate codons identified by SwiftLib [16]. All nine models are considered when choosing amino acids to encode,

in case some models are capturing different characteristics of the sequence-function relationship. While the optimal combinations of amino acids identified by the model are retained in this library, there may be non-optimal combinations that result from this procedure. However, we have developed this method to balance these experimental costs with being able to access the restricted libraries. The degenerate codons used to encode the predicted libraries are shown in Figure 2.8. The predicted libraries were tested in the same manner as above.

2.6 Supplemental Information

Sample prediction frequency table – position Set II (*S*)-

A sample predicted frequency table from position Set II for the (*S*)-enantiomer is provided below in Table 2.3. At this step, the exact combinations predicted by the machine learning models are lost and instead interpreted as frequencies at individual amino acid positions to encode with degenerate codons. The alternative (ordering and cloning the top sequences individually) can be quite expensive, but tractable if screening costs significantly outweigh DNA synthesis costs. All 20 canonical amino acids are enumerated, although each library does not contain all 20 in the input library. Amino acids that are not present in the input library, but predicted to be high functioning, can be used as indicators for when the frequencies may be relying on amino acids at other positions to make predictions. In other words, they serve as cut-offs above which the amino acids should be considered.

Table 2.3: Sample Prediction Frequency Table

AA1	Freq1	AA2	Freq2	AA3	Freq3
Y	153	V	203	I	302
N	115	F	123	V	288
R	91	I	80	L	93
G	81	Y	54	S	73
Q	50	Q	48	F	51
S	49	W	46	M	27
T	47	H	45	Q	21
W	47	M	43	K	19
K	47	L	43	T	19
E	46	A	43	P	19
A	45	E	40	A	18
H	37	K	40	W	17
V	37	T	37	E	13
M	36	R	36	G	9
C	35	P	36	H	9
D	31	S	32	Y	7
F	21	C	27	N	6
L	12	N	10	C	5
P	11	G	8	D	2
I	9	D	6	R	2

For example, the first amino acids that were not present in the input dataset for each of the three positions are Q, Q, M as NDT encodes: {N, S, I, H, R, L, D, G, V, Y, C, F}. The amino acids occurring significantly more frequently in the top 1000 sequences

are then [Y, N, R, G], [V, F, I], and [I, V, L, S, F]. This process is repeated for the top three models determined by default hyperparameters, and then 3 hyperparameter sets are used for each optimal, for a total of nine models (in an attempt to capture different portions of the landscape with inaccurate models).

In the described approach, this step can be tuned to consider more or less sequences (such as the top 20% or top 1%) depending on the protein engineer's discretion considering the following: screening throughput, sequencing cost, cloning capabilities, desired fitness improvement, model accuracy, theoretical size of the predicted library, ease of encoding with codon degeneracy, and an interpretation of the landscape (how many variants are expected to be near the fitness peak). As DNA synthesis costs continue to fall, the ideal is to be able to sample the top sequences directly (as was simulated in Chapter 3 with data from a full recombination library of four positions) without resorting to this approach to interpret the models' predictions with degenerate codons.

Variant Starting Activity

Although WT has slightly lower enantioselectivity (and thus may reach both enantiomers more easily), we started with a previously engineered variant, Y32K V97L, for its significantly higher activity, which we hypothesized would make data collection more reproducible. Activity and selectivity are reported in biological triplicate.

Table 2.4: Summary of starting activity observed in *Rma* NOD variants.

Excess	(*S*)-enantiomer Area (mAU*s)	(*R*)-enantiomer Area (mAU*s)	Enantiomeric Excess
Rma NOD WT	1350 ± 90	340 ± 30	59 %
Rma NOD Y32K V97L	2710 ± 50	370 ± 20	76%

Modeling Statistics for *Rma* NOD

The accuracy for the nine models of each Set, as well as the average values of the nine models, is shown for the data obtained by screening the predicted libraries. The corresponding predicted versus measured values can be found in the following section for Set I (Table 2.5), Set IIR (Table 2.6), and Set IIS (Table 2.7).

Table 2.5: Test errors for Set I from predicted (*R*)- and (*S*)- libraries

Position Set I	Kendall tau	MAE	Pearson r
Model_0	0.32518	30.01188	0.480996
Model_1	0.32518	30.0119	0.480996
Model_2	0.32518	30.01191	0.480996
Model_3	0.382935	22.62033	0.471391
Model_4	0.452322	27.23348	0.548192
Model_5	0.347679	32.51843	0.428702
Model_6	0.362329	30.58998	0.514802
Model_7	0.365468	30.38801	0.516186
Model_8	0.404083	17.22301	0.508073
Average	0.390582	26.07764	0.517561

Table 2.6: Test errors for Set II from predicted (*R*)- library

Position Set II (*R*)-	Kendall tau	MAE	Pearson r
Model_0	0.608254	0.44737	0.822691
Model_1	0.608254	0.447369	0.822691
Model_2	0.595273	0.446588	0.818083
Model_3	0.28028	0.223076	0.385406
Model_4	0.626798	0.194871	0.827613
Model_5	0.626798	0.196244	0.821946
Model_6	0.598982	0.410763	0.823711
Model_7	0.598982	0.410763	0.823711
Model_8	0.598982	0.410763	0.823711
Average	0.610108	0.285607	0.820453

Table 2.7: Test errors for Set II from predicted (*S*)- library

Position Set II (*S*)-	**Kendall tau**	**MAE**	**Pearson r**
Model_0	-0.03104	0.230254	-0.04925
Model_1	0.020243	0.209426	0.072554
Model_2	0.041835	0.216491	0.098214
Model_3	0.22807	0.109102	0.342213
Model_4	0.265857	0.123463	0.363029
Model_5	0.063428	0.139915	0.118149
Model_6	0.086663	0.215941	0.144677
Model_7	0.086663	0.215941	0.144677
Model_8	0.086663	0.215941	0.144677
Average	0.060729	0.1903	0.108533

Activity of Final Variants compared to Starting

The relative activity compared to KFLL is shown for the top 3 variants from each input and predicted round.

Table 2.8: Relative activity compared to starting sequence - Set I

Set I: Residues 32, 46, 56, and 97

	Input Variants					Input Variants				
	Residue				Activity Rel to KFLL	Residue				Activity Rel to KFLL
	32	46	56	97		32	46	56	97	
	Y	N	L	L	2.0	V	G	V	L	1.9
(*S*)-	C	S	V	L	1.7	C	F	N	L	2.1
	C	V	H	V	2.4	V	C	H	V	2.5
	C	R	S	G	2.2	G	S	S	G	2.7
(*R*)-	I	S	C	G	2.0	G	F	L	R	1.1
	N	V	R	I	2.3	H	C	S	R	0.9

Table 2.9: Relative activity compared to starting sequence - Set II

Set II: Residues 49, 51, and 53

	Input Variants						Predicted Variants					
	Residue			Enantioselectivity		Activity Rel.	Residue			Enantioselectivity		Activity Rel.
	49	51	53	% S-isomer	% R-isomer	to KFLL	49	51	53	% S-isomer	% R-isomer	to KFLL
(*S*)-selective	P	R	I	93	7	2.5	Y	V	V	97	3	2.8-fold
from	Y	V	F	93	7	1.5	P	V	I	96	4	3.2-fold
VCHV	N	D	V	87	13	0.8	P	V	V	96	4	3.1-fold
(*R*)-selective	P	R	I	19	81	2.7	P	R	L	11	89	2.2-fold
From	Y	F	F	22	78	0.8	P	G	L	13	87	2.1-fold
GSSG	C	V	N	24	76	0.6	P	F	F	15	85	2.2-fold

Predicted vs Measured Values for All Libraries

The predicted versus measured values for sequence-verified variants in the predicted libraries are shown for each library. Figure 2.5 contains predicted values for position Set I. Figure 2.6 contains predicted values for position Set II from GSSG, and Figure 2.7 for Set II from VCHV. A linear regression is shown for these values.

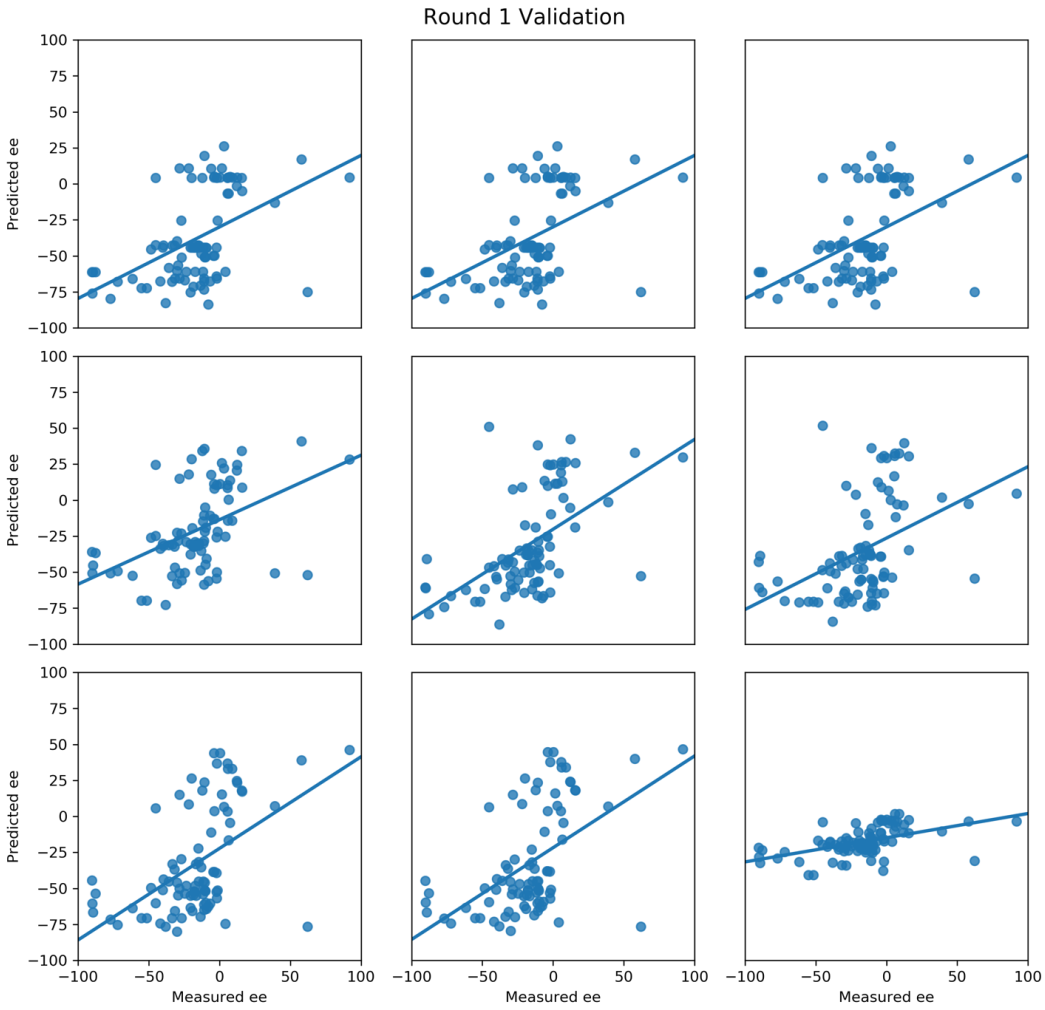


Figure 2.5: Predicted vs measured values for *ee* from Set I.

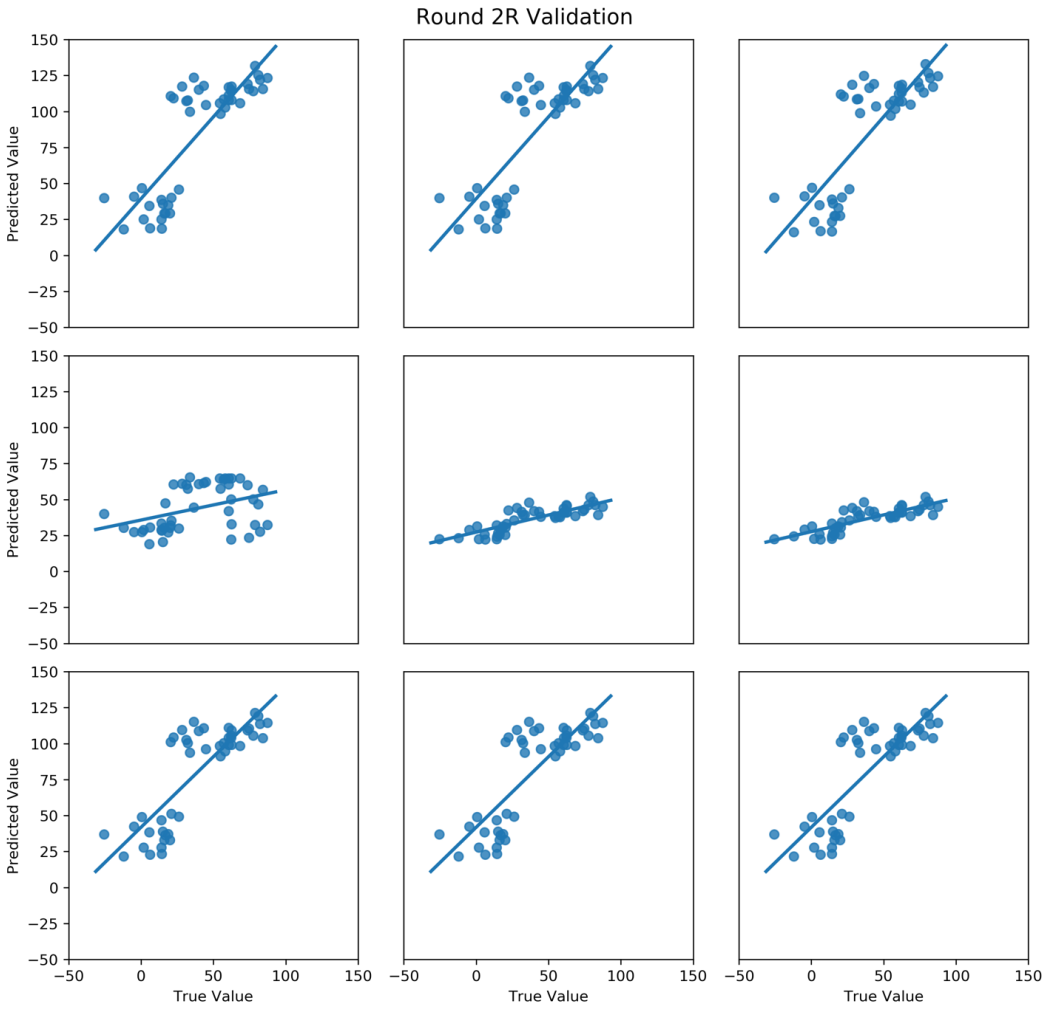


Figure 2.6: Predicted vs measured values for *ee* from position Set II from GSSG.

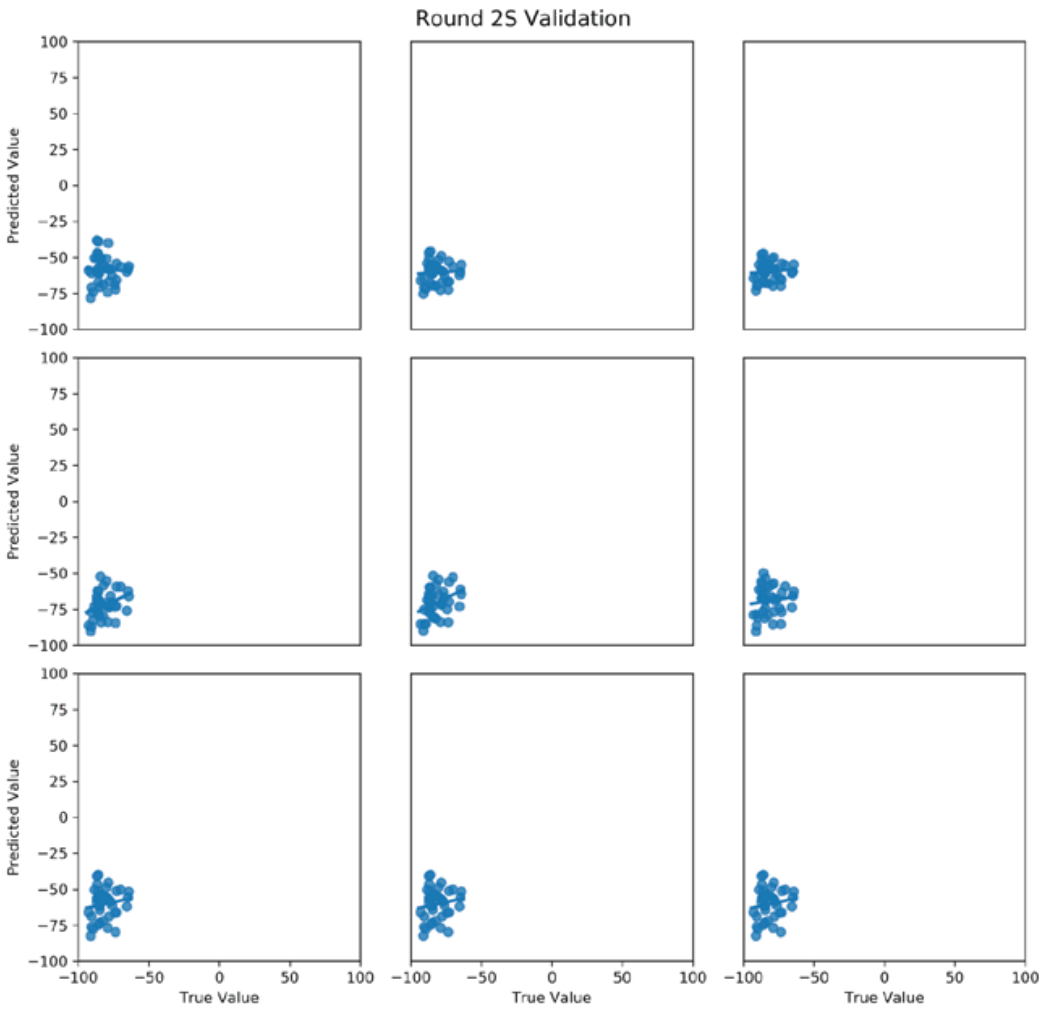
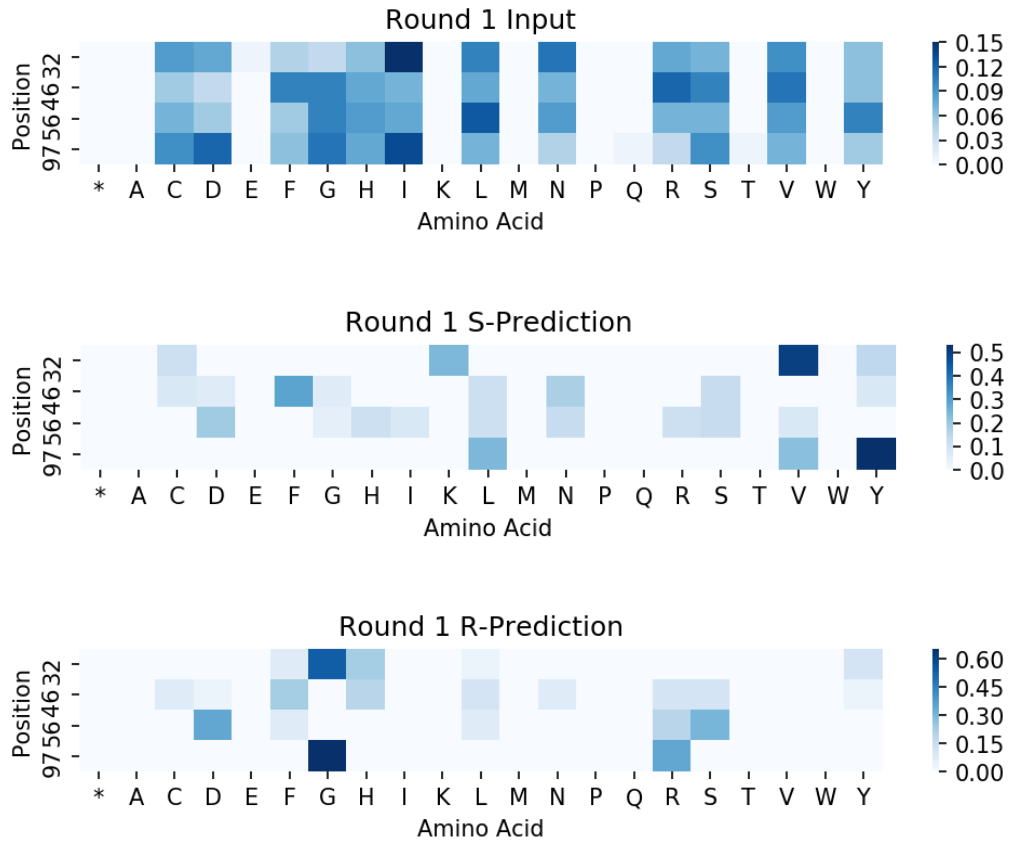


Figure 2.7: Predicted vs Measured Values for *ee* from Position Set II from VCHV.

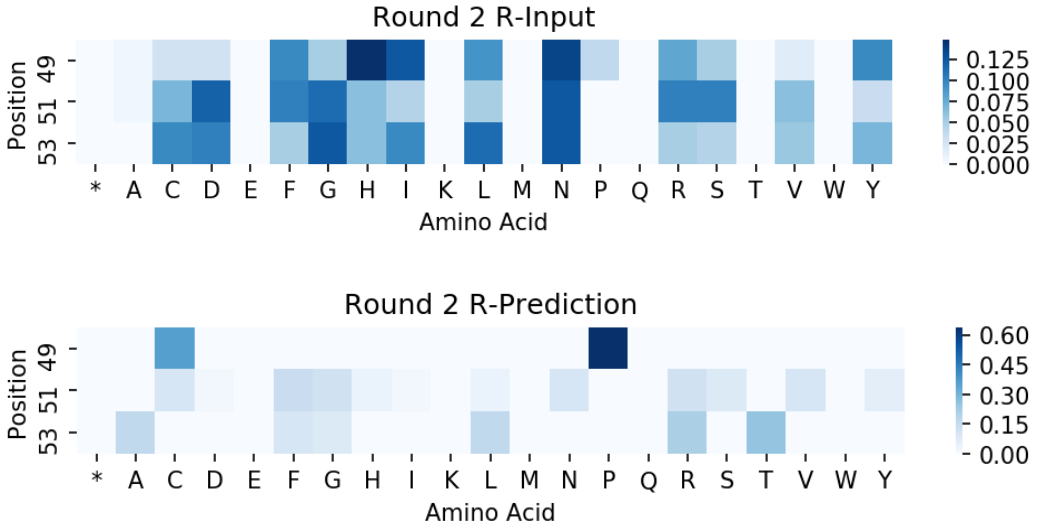
Input sequences versus encoded predictions

To verify that the distribution of amino acids in the predicted libraries differs from that of the input, heat maps of encoded amino acids are shown for each round comparing the two. (These ratios are often represented with sequence logo maps, which are better visualizations when a few amino acids dominate.) The input libraries are NDT libraries, which represent N, S, I, H, R, L, D, G, V, Y, C, F. Input libraries also contain proline at position 49 from WT. The degenerate codons used to encode amino acids at each position are provided for reference. Sequence-function data is available on Protabank [44]. In these tables, residues in bold were not part of the predicted library but had to be included with the degenerate codon cloning method.



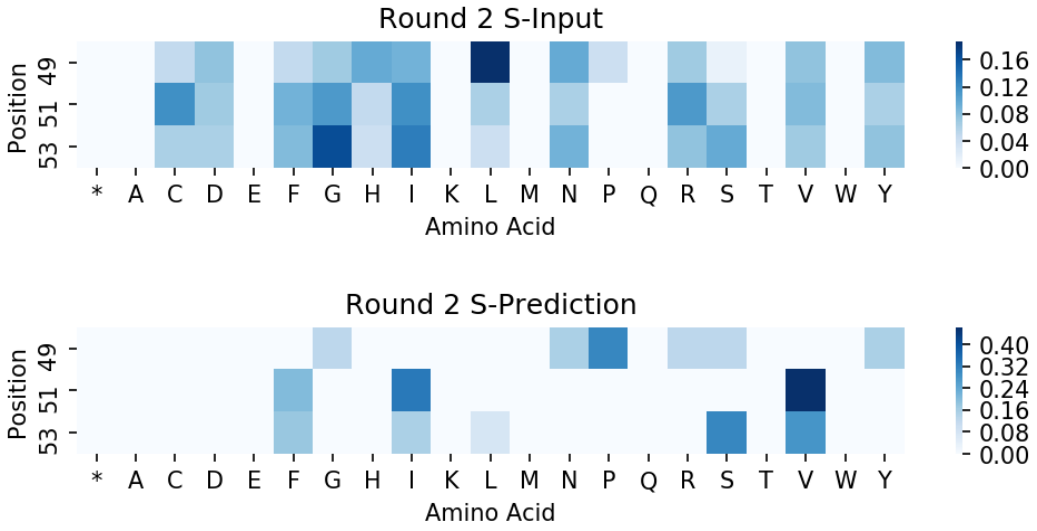
1R Predictions			1S Predictions		
Position	Codon	Encoded	Position	Codon	Encoded
32	AAA; GTA; TRC	K; V; C, Y	32	GGA; YWC	G; F, H, L, **Y**
46	DRC; TTM	C, D, G, N, S, Y; F, L	46	HDC	C, F, H, I, L, N, **R**, S, Y
56	VDC	**D**, G, H, I, L, N, R, S, V	56	GAC; YBC	D; C, F, L, **P**, R, S
97	STA; TAC	L, V; Y	97	RGA	G, R

Figure 2.8: Input versus predicted sequences for modeling position Set I.



Position	Codon	Encoded
49	CCA; TGC	P, C
51	NDT	F, V, Y, N, R, G, I, **H**, **L**, **D**, **C, S**
53	RSA; TTM	L, F, G, R, **T**; S, C

Figure 2.9: Input versus predicted sequences for modeling Set II from GSSG.



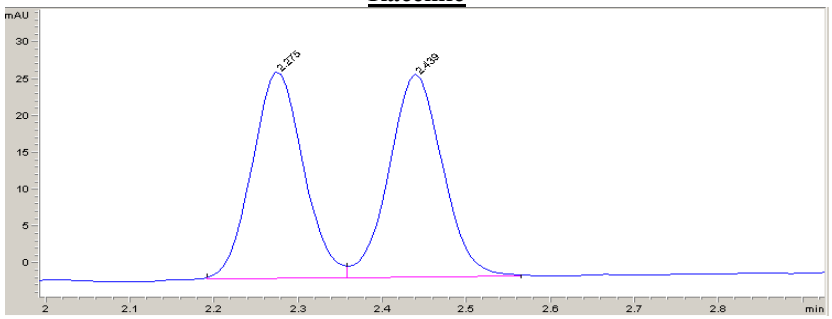
Position	Codon	Encoded
49	HMC; RGA	Y, N, R; S, P, G
51	DTC	V, F, I
53	AGC; NTC	I, V, L; S, F

Figure 2.10: Input versus predicted sequences for modeling Set II from VCHV.

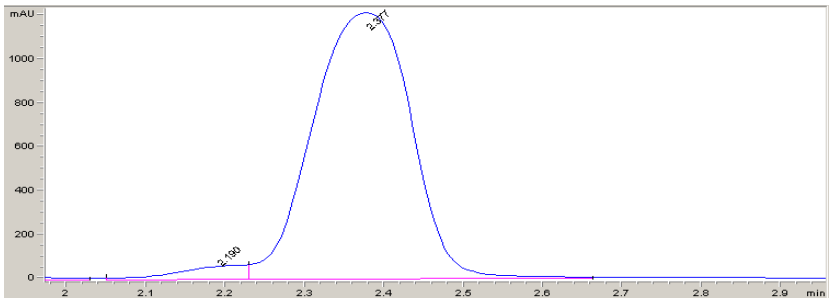
Chiral SFC traces for racemic and enzymatically synthesized organosilicon products

Chiralcel OD-H (4.6 mm x 25 cm), 5% isopropanol in CO_2, 3 mL/min, 210 nm

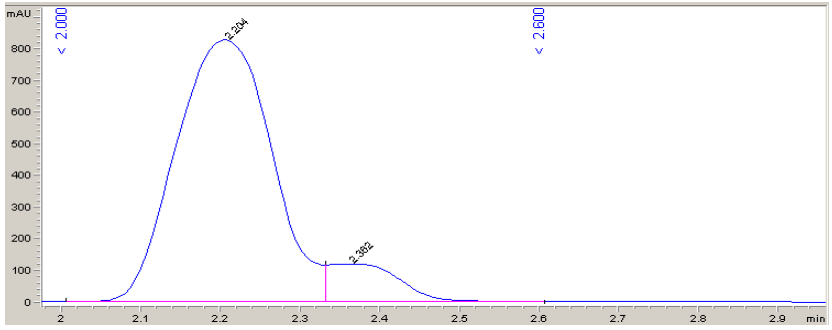
Racemic



Variant VCHVYVV



Variant GSSGPRL



rac			**VCHVYVV**			**GSSGPRL**		
Retention Time (min)	Area (mAU*S)	Area %	Retention Time (min)	Area (mAU*S)	Area %	Retention Time (min)	Area (mAU*S)	Area %
2.275	119.2	49.4%	2.19	418.3	3.8%	2.204	6903.2	90.4%
2.439	121.9	50.6%	2.377	10498.3	96.2%	2.362	733.6	9.6%
Total	241.1		Total	10916.6		Total	7636.8	

Figure 2.11: Summary of Chiral SFC trace. All the *ee* values of synthesized organosilicon products were determined using automatic peak integration from chiral SFC. The traces for racemic and enzymatic products are shown with summarized integration.

Experimental uncertainty in best *Rma* NOD variants

Although the protein GB1 case study is presented as proof of principle, we also provide evidence that this approach results in significantly improved variants over the input proteins for Position Set II. However, we would like to reiterate that while we have shown this method is more likely to find better variants on an empirical method, this method does not guarantee identifying protein variants that are better than the best identified in the input library. A simple case example is serendipitously identifying the fitness maximum in the input library. The p-values obtained from Welch's t-test are shown below.

Table 2.10: Activity is significantly improved over starting variant KFLL

Variant	Mean ± StDev	p-value
KFLLPRI	3290 ± 360	—
VCHVYVV	9330 ± 1780	3.77E-02
VCHVPVI	10670 ± 520	2.26E-04
VCHVPVV	7820 ± 1820	6.84E-04
GSSGPRL	7380 ± 190	2.27E-05
GSSGPGL	7020 ± 230	3.37E-05
GSSGPFF	7300 ± 440	6.06E-04

Comparisons for enantioselectivity are best done with a different metric than what is typically reported (*ee*). Enantiomeric excess refers to the positive ratio of the following: $\frac{|R-S|}{(R+S)}$. A key assumption of the t-test is that each population has a normal distribution, therefore we first convert *ee* to $\Delta\Delta G$ by taking $\ln(R/S)$ where R is the major product or $\ln(S/R)$ in the opposite case.

Table 2.11: Enantioselectivity in Set II is significantly improved over starting variant GSSG

Variant	Mean ± StDev of ln(S/R)	p-value
GSSGPRI	1.484 ± 0.090	—
GSSGPRL	2.152 ± 0.063	1.65E-03
GSSGPGL	1.925 ± 0.034	1.21E-02
GSSGPFF	1.731 ± 0.062	3.93E-02

Table 2.12: Enantioselectivity in Set II is significantly improved over starting variant VCHV

Variant	Mean ± StDev of ln(R/S)	p-value
VCHVPRI	2.596 ± 0.070	—
VCHVYVV	3.386 ± 0.103	1.48E-03
VCHVPVI	3.213 ± 0.152	1.62E-03
VCHVPVV	3.128 ± 0.010	7.54E-03

Model performance and selection

In the first pass for model selection, the LOO Pearson r of the regressors with default hyperparameters are used. The models ultimately selected are subsequently shown for each round.

Set I Initial Models

0.512212499 GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

0.478097911 RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

0.460760125 LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000, random_state=None, tol=0.0001, verbose=0)

0.447166856 ARDRegression(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.001, verbose=False)

0.423793421 KernelRidge(alpha=1, coef0=1, degree=3, gamma=None, kernel='linear', kernel_params=None)

0.419172462 BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, tol=0.001, verbose=False)

0.406655665 BaggingRegressor(base_estimator=None, bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

0.396791771 LassoLarsCV(copy_X=True, cv=None, eps=2.2204460492503131e-16, fit_intercept=True, max_iter=500, max_n_alphas=1000, n_jobs=1, normalize=True, positive=False, precompute='auto', verbose=False)

0.37899373 DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

0.371734032 SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01, fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling', loss='squared_loss', n_iter=5, penalty='l2', power_t=0.25, random_state=None, shuffle=True, verbose=0, warm_start=False)

0.366256085 KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')

0.338423931 ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5, max_iter=1000, normalize=False, positive=False, precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False)

0.202908183 AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear', n_estimators=50, random_state=None)

-0.082371549 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

-0.766587492 NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False)

Set I Selected Models

ARDRegression(alpha_1=0.01, alpha_2=0.1, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.01, lambda_2=0.01, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

ARDRegression(alpha_1=0.01, alpha_2=0.01, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.01, lambda_2=0.01, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

ARDRegression(alpha_1=0.01, alpha_2=0.0001, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.01, lambda_2=0.01, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

GradientBoostingRegressor(alpha=0.3, criterion='mse', init=None,learning_rate=0.9, loss='quantile', max_depth=3,max_features=None, max_leaf_nodes=None,min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0,n_estimators=500, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

GradientBoostingRegressor(alpha=0.5, criterion='mse', init=None,learning_rate=0.7, loss='huber', max_depth=10,max_features=None, max_leaf_nodes=None,min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0,n_estimators=1000, presort='auto', random_state=None,subsample=1.0, verbose=0, warm_start=False)

GradientBoostingRegressor(alpha=0.5, criterion='mse', init=None,learning_rate=0.7, loss='huber', max_depth=10,max_features=None, max_leaf_nodes=None,min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0,n_estimators=100, presort='auto', random_state=None,subsample=1.0, verbose=0, warm_start=False)

LinearSVR(C=50, dual=True, epsilon=0, fit_intercept=True,intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000,random_state=None, tol=0.0001, verbose=0)

LinearSVR(C=50, dual=True, epsilon=0.1, fit_intercept=True,intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000,random_state=None, tol=0.0001, verbose=0)

LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True,intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000,random_state=None, tol=0.0001, verbose=0)

Set IIS Initial Models

0.609011 ARDRegression(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.001, verbose=False)

0.60823 NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False)

0.598652 LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000, random_state=None, tol=0.0001, verbose=0)

0.587369 KernelRidge(alpha=1, coef0=1, degree=3, gamma=None, kernel='linear', kernel_params=None)

0.584004 BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, tol=0.001, verbose=False)

0.578776 GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

0.577483 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

0.564136 BaggingRegressor(base_estimator=None, bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

0.549607 RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

0.503091 MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)

0.499812 DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

0.438121 LassoLarsCV(copy_X=True, cv=None, eps=2.2204460492503131e-16, fit_intercept=True, max_iter=500, max_n_alphas=1000, n_jobs=1, normalize=True, positive=False, precompute='auto', verbose=False)

0.437293 SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01, fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling', loss='squared_loss', max_iter=None, n_iter=None, penalty='l2', power_t=0.25, random_state=None, shuffle=True, tol=None, verbose=0, warm_start=False)

0.430372 KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')

0.399228 AdaBoostRegressor(base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=None, loss='linear')

Set IIS Selected Models

GradientBoostingRegressor(alpha=0.1, criterion='mse', init=None, learning_rate=0.7, loss='lad', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

GradientBoostingRegressor(alpha=0.5, criterion='friedman_mse', init=None, learning_rate=0.9, loss='quantile', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

GradientBoostingRegressor(alpha=0.3, criterion='friedman_mse', init=None, learning_rate=0.3, loss='quantile', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

ARDRegression(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.1, lambda_2=1e-06, n_iter=3000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

ARDRegression(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.1, lambda_2=1e-06, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

ARDRegression(alpha_1=0.1, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.1, lambda_2=1e-06, n_iter=3000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

LinearSVR(C=50, dual=True, epsilon=0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000, random_state=None, tol=0.0001, verbose=0)

LinearSVR(C=100, dual=True, epsilon=0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000, random_state=None, tol=0.0001, verbose=0)

LinearSVR(C=1000, dual=True, epsilon=0, fit_intercept=True, intercept_scaling=1.0, loss='squared_epsilon_insensitive', max_iter=10000, random_state=None, tol=0.0001, verbose=0)

Set IIR Initial Models

0.651306 GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

0.635081 ARDRegression(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.001, verbose=False)

0.631197 BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, tol=0.001, verbose=False)

0.626982 KernelRidge(alpha=1, coef0=1, degree=3, gamma=None, kernel='linear', kernel_params=None)

0.625465 AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear', n_estimators=50, random_state=None)

0.61362 NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False)

0.612285 LassoLarsCV(copy_X=True, cv=None, eps=2.2204460492503131e-16, fit_intercept=True, max_iter=500, max_n_alphas=1000, n_jobs=1, normalize=True, positive=False, precompute='auto', verbose=False)

0.608155 RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

0.595166 MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)

0.583177 BaggingRegressor(base_estimator=None, bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

0.552808 LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000, random_state=None, tol=0.0001, verbose=0)

0.542432 SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01, fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling', loss='squared_loss', max_iter=None, n_iter=None, penalty='l2', power_t=0.25, random_state=None, shuffle=True, tol=None, verbose=0, warm_start=False)

0.479498 DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

0.473718 KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')

0.242713 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

Set IIR Selected Models

GradientBoostingRegressor(alpha=0.1, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

GradientBoostingRegressor(alpha=0.5, criterion='mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=500, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

GradientBoostingRegressor(alpha=0.7, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)

ARDRegression(alpha_1=1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=10000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

ARDRegression(alpha_1=1, alpha_2=0.0001, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=10000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

ARDRegression(alpha_1=1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.0001, lambda_2=0.1, n_iter=10000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)

BayesianRidge(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=10000, normalize=False, tol=0.0001, verbose=False)

BayesianRidge(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=3000, normalize=False, tol=0.0001, verbose=False)

BayesianRidge(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=300, normalize=False, tol=0.0001, verbose=False)

2.7 Bibliography

References

1. Dušan, P. & Lynn, K. S. C. Molecular modeling of conformational dynamics and its role in enzyme evolution. *Current Opinion in Structural Biology* **52,** 50–57 (2018).

2. Romero, P. A. & Arnold, F. H. Exploring protein fitness landscapes by directed evolution. *Nature Reviews Molecular Cell Biology* **10,** 866–876. doi:10.1038/nrm2805 (2009).

3. Goldsmith, M. & Tawfik, D. S. Enzyme engineering: reaching the maximal catalytic efficiency peak. *Current Opinion in Structural Biology* **47,** 140–150 (2017).

4. Zeymer, C. & Hilvert, D. Directed evolution of protein catalysts. *Annual Review of Biochemistry* **87,** 131–157 (2018).

5. Garcia-Borrás, M., Houk, K. N. & Jiménez-Osés, G. Computational design of protein function. *Computational Tools for Chemical Biology* **3,** 87. doi:10.1039/9781788010139-00087 (2017).

6. Lewis, R. D. *et al.* Catalytic iron-carbene intermediate revealed in a cytochrome c carbene transferase. *Proceedings of the National Academy of Sciences USA* **115,** 7308–7313 (2018).

7. Dahiyat, B. I. & Mayo, S. L. De novo protein design: fully automated sequence selection. *Science* **278,** 82–87 (1997).

8. Khersonsky, O. *et al.* Automated design of efficient and functionally diverse enzyme repertoires. *Molecular Cell* **72,** 178–186 (2018).

9. Amrein, B. A. *et al.* CADEE: Computer-aided directed evolution of enzymes. *IUCrJ* **4,** 50–64 (2017).

10. Murphy, K. P. *Machine learning: a probabilistic perspective* (MIT press, 2012).

11. Jordan, M. I. & Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science* **349,** 255–260 (2015).

12. Silver, D. *et al.* Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv* **550,** 354–359 (2017).

13. Fox, R. J. *et al.* Improving catalytic function by ProSAR-driven enzyme evolution. *Nature Biotechnology* **25,** 338 (2007).

14. Romero, P. A., Krause, A. & Arnold, F. H. Navigating the protein fitness landscape with Gaussian processes. *Proceedings of the National Academy of Sciences USA* **110,** e193–e201. doi:10.1073/pnas.1215251110 (2013).

15. Yang, K. K., Wu, Z. & Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nature Methods* **16,** 687–694. doi:10.1038/s41592-019-0496-6 (2019).

16. Jacobs, T. M., Yumerefendi, H., Kuhlman, B. & Leaver-Fay, A. SwiftLib: rapid degenerate-codon-library optimization through dynamic programming. *Nucleic Acids Research* **43,** e34–e34 (2015).

17. Kan, S. J., Lewis, R. D., Chen, K. & Arnold, F. H. Directed evolution of cytochrome c for carbon–silicon bond formation: Bringing silicon to life. *Science* **354,** 1048–1051 (2016).

18. Showell, G. A. & Mills, J. S. Chemistry challenges in lead optimization: silicon isosteres in drug discovery. *Drug Discovery Today* **8,** 551–556 (2003).

19. Franz, A. K. & Wilson, S. O. Organosilicon molecules with medicinal applications. *Journal of Medicinal Chemistry* **56,** 388–405 (2013).

20. Shi, S.-L., Wong, Z. L. & Buchwald, S. L. Copper-catalysed enantioselective stereodivergent synthesis of amino alcohols. *Nature* **532,** 353–356 (2016).

21. Finefield, J. M., Sherman, D. H., Kreitman, M. & Williams, R. M. Enantiomeric natural products: occurrence and biogenesis. *Angewandte Chemie International Edition* **51,** 4802–4836 (2012).

22. Reetz, M. T. Controlling the enantioselectivity of enzymes by directed evolution: practical and theoretical ramifications. *Proceedings of the National Academy of Sciences USA* **101,** 5716–5722 (2004).

23. Consortium, U. UniProt: the universal protein knowledgebase. *Nucleic Acids Research* **45,** d158–d169. doi:10.1093/nar/gkw1099 (2017).

24. Bloom, J. D., Labthavikul, S. T., Otey, C. R. & Arnold, F. H. Protein stability promotes evolvability. *Proceedings of the National Academy of Sciences USA* **103,** 5869–5874 (2006).

25. Waterhouse, A. *et al.* SWISS-MODEL: homology modelling of protein structures and complexes. *Nucleic Acids Research* **46,** W296–w303 (2018).

26. Fox, R. J. *et al.* Optimizing the search algorithm for protein engineering by directed evolution. *Protein Engineering* **16,** 589–597 (2003).

27. Kille, S. *et al.* Reducing codon redundancy and screening effort of combinatorial protein libraries created by saturation mutagenesis. *ACS Synthetic Biology* **2,** 83–92 (2013).

28. Lissy, N. A. Patentability of chemical and biotechnology inventions: A discrepancy in standards. *Washington University of Law Quarterly* **81,** 1069 (2003).

29. Steinberg, B. & Ostermeier, M. Environmental changes bridge evolutionary valleys. *Science Advances* **2,** e1500921 (2016).

30. Drummond, D. A., Iverson, B. L., Georgiou, G. & Arnold, F. H. Why high-error-rate random mutagenesis libraries are enriched in functional and improved proteins. *Journal of Molecular Biology* **350,** 806–816 (2005).

31. Wu, N. C., Dai, L., Olson, C. A., Lloyd-Smith, J. O. & Sun, R. Adaptation in protein fitness landscapes is facilitated by indirect paths. *Elife* **5,** e16965. doi:10.7554/eLife.16965 (2016).

32. Bershtein, S., Segal, M., Bekerman, R., Tokuriki, N. & Tawfik, D. S. Robustness–epistasis link shapes the fitness landscape of a randomly drifting protein. *Nature* **444,** 929–932 (2006).

33. Zhang, R. K. *et al.* Enzymatic assembly of carbon–carbon bonds via iron-catalysed sp 3 C–H functionalization. *Nature* **565,** 67–72 (2019).

34. Kim, D. E., DiMaio, F., Yu-Ruei Wang, R., Song, Y. & Baker, D. One contact for every twelve residues allows robust and accurate topology-level protein structure modeling. *Proteins: Structure, Function, and Bioinformatics* **82,** 208–218 (2014).

35. Fowler, D. M. & Fields, S. Deep mutational scanning: a new style of protein science. *Nature Methods* **11,** 801. doi:10.1038/nmeth.3027 (2014).

36. Sinai, S., Kelsic, E., Church, G. M. & Nowak, M. A. Variational auto-encoding of protein sequences. *arXiv* (2017).

37. Riesselman, A. J., Ingraham, J. B. & Marks, D. S. Deep generative models of genetic variation capture the effects of mutations. *Nature Methods* **15,** 816–822 (2018).

38. Brookes, D. H. & Listgarten, J. Design by adaptive sampling. *arXiv* (2018).

39. Saito, Y. *et al.* Machine-Learning-Guided Mutagenesis for Directed Evolution of Fluorescent Proteins. *ACS Synthetic Biology* **7,** 2014–2022. doi:10.1021/acssynbio.8b00155 (2018).

40. Cadet, F. *et al.* A machine learning approach for reliable prediction of amino acid interactions and its application in the directed evolution of enantioselective enzymes. *Scientific Reports* **8,** 1–15 (2018).

41. Carlin, D. A. *et al.* Kinetic characterization of 100 glycoside hydrolase mutants enables the discovery of structural features correlated with kinetic constants. *PloS One* **11,** e0147596. doi:10.1371/journal.pone.0147596 (2016).

42. Barley, M. H., Turner, N. J. & Goodacre, R. Improved descriptors for the quantitative structure–activity relationship modeling of peptides and proteins. *Journal of Chemical Information and Modeling* **58,** 234–243. doi:10.1021/acs.jcim.7b00488 (2018).

43. Yang, K. K., Wu, Z., Bedbrook, C. N. & Arnold, F. H. Learned protein embeddings for machine learning. *Bioinformatics* **34,** 2642–2648. doi:10.1093/bioinformatics/bty178 (2018).

44. Wang, C. Y. *et al.* ProtaBank: A repository for protein design and engineering data. *Protein Science* **27,** 1113–1124. doi:10.1002/pro.3406 (2018).

45. Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830 (2011).

46. Gibson, D. G. *et al.* Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nature Methods* **6,** 343–345 (2009).

Chapter 3

COMPARING EVOLUTIONARY STRATEGIES ON AN EMPIRICAL AND EPISTATIC FITNESS LANDSCAPE

1. Wu, Z., Kan, S. B. J., Lewis, R. D., Wittmann, B. J. & Arnold, F. H. Machine learning-assisted directed protein evolution with combinatorial libraries. *Proceedings of the National Academy of Sciences USA* (2019).

Contributions Statement: Z.W. performed all experiments and data analyses.

3.1 Abstract

Machine learning is able to guide directed evolution to enriched regions of the protein fitness landscape. However, directed evolution on its own is similarly capable of reaching local optima in these landscapes. In this chapter, we demonstrate that machine learning-guided directed evolution finds variants with higher fitness than those found by other directed evolution approaches, even when using naïve implementations of machine learning. This approach is validated on a large, previously published empirical fitness landscape for human GB1 binding protein.

3.2 Introduction

Directed evolution techniques have enabled the adaptation of natural sequences to human applications [1, 2]. By relying on data generated for desired tasks, and less so on biophysical models and computation, directed evolution has been able to circumvent our inability to accurately model physical processes and enable engineering of any measurable property. However, the major bottleneck in directed evolution is the experimental burden: testing hundreds to thousands of variants remains the rate limiting step in protein engineering. While biology is fortunate to have Moore's Law applied to both DNA synthesis [3] and sequencing [4], experimentation is unlikely to follow on the same trajectory, although improvements to experimental work flow and screening throughput optimization are being made constantly [5].

Limited by screening capacity, most traditional directed evolution campaigns rely on the identification and accumulation of beneficial single mutations, where single mutations are defined as mutations to specific amino acids at specific positions [6].

In these approaches, these single mutations are identified in greedy walks through Maynard Smith's proverbial protein fitness landscape [7], with the assumption that accumulating these single mutations is generally safe. This approach works particularly well in smooth landscapes, where the contributions of mutations are additive such that accumulating beneficial single mutations allows for steady climbs continuously to local optima.

However, it is well established that proteins are epistatic molecules [8, 9]. In other words, there are non-additive contributions to protein fitness for each amino acid. Unfortunately, current screening capabilities do not allow directly sampling the combinatorial epistatic landscape, where the number of possible sequences grows with 20^N, N being the number of positions under consideration. Instead, protein engineers are restricted to exploring single mutations, where the number of sequences scales with $20N$. While this approach reduces the amount of experimental burden, it does not adequately sample epistatic landscapes.

To address this limitation, we have developed a method incorporating machine learning to steer directed evolution campaigns by leveraging all of the generated experimental data (Figure 3.1 bottom), where previously, only the best variants were used (Figure 3.1 top). I outlined the machine learning-assisted strategy outlined in Chapter 2, and give further validation here: in this approach, multiple amino acid residues are randomized in each generation, enabling sampling of the epistatic landscape. Sequence-function relationships are then used to train a predictive algorithm, which is tasked with performing a round of *in silico* evolution. Thus, the experimental burden of combinatorial landscapes, which is too high for physical exploration, is moved to computational resources.

In Chapter 2, we applied this approach to the evolution of an enzyme without prior knowledge of the landscape. In this chapter, we test this approach against others on an empirical fitness landscape obtained for protein binding [10] obtained by deep mutational scanning [11]. On this landscape, we show that a naïve implementation of this method is sufficient to outperform other methods by reaching the global fitness optimum at a higher rate (8.2%, compared to 4.9% in the next best method). It also offers a higher expected fitness from the evolutionary campaign (6.42, compared to 5.93 in the next best method), given equal experimental burden. While this evidence is empirical and limited to one protein fitness landscape, we have limited the machine learning approach to provide a (low) base comparison to existing evolutionary methods and expect many improvements to come.

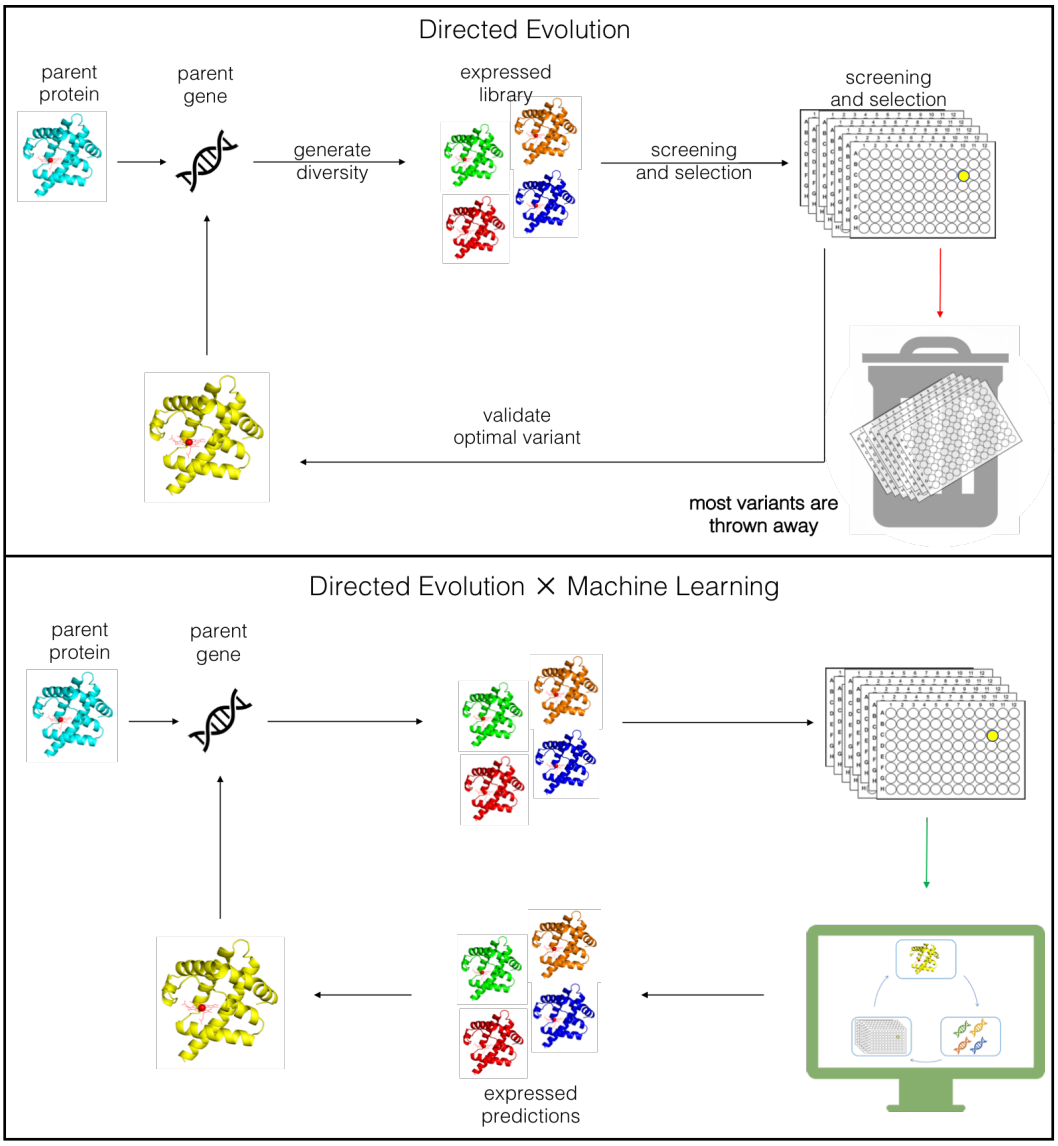


Figure 3.1: Top: Traditional directed evolution. After experimentation, only the best variant(s) are kept for future rounds of evolution. Bottom: Machine Learning-Assisted Directed Evolution. Information from all experiments is used to train a supervised machine learning model and used to identify the best variant(s) in a round of *in silico* testing

.

3.3 Results

Evolutionary Approaches to Protein Engineering

To validate the developed machine learning-assisted approach, we compare it to other evolutionary methods on the large empirical fitness landscape obtained on a protein G domain B1(GB1) binding to an antibody [10]. Specifically, we compare

the final fitnesses reached by simulated directed evolution with and without machine learning, based on testing the same number of variants. The empirical landscape used here consists of measurements of 149,361 out of a total $20^4 = 160,000$ variants from NNK/NNS saturation-mutagenesis libraries at four positions known to interact epistatically. The fitness of protein GB1 was defined as the enrichment of folded protein bound to the antibody IgG-Fc, measured by coupling mRNA display with next-generation sequencing and normalized to the parent sequence, which had a fitness value of 1.0. The landscape contains a fitness maximum at 8.76, and 19.7% of variants at a reported value of 0 (dead variants). On this landscape, the simulated single-mutant walk (described below) reached 869 local optima, 533 of which outperformed the wild-type sequence and 138 of which had fitness less than 5% of the wild-type fitness. For a full description of the epistatic landscape, see the thorough analysis of Wu and coworkers [10].

We first simulated single-mutation evolutionary walks starting from each of the 149,361 variants reported. The algorithm proceeded as follows:

1. In each single-mutation walk, all possible single amino acid mutations were tested at each of the four mutated positions.
2. The best amino acid was then fixed at its observed position, and that position was restricted from further exploration.
3. Repeat the first two steps with the remaining positions until an amino acid is fixed at each position.

As a greedy search algorithm that always follows the path with strongest improvements in fitness, this single mutation walk has a deterministic solution for each starting variant. Assuming each amino acid occurs with equal frequency and that the library has complete coverage, applying the 3-fold oversampling rule to obtain roughly 95% library coverage [12, 13] results in a total of 570 variants screened. We provide further explanation for this number in the following section.

Another technique widely used in directed evolution is recombination. For a given set of positions to explore, one method is to randomly sample the combinatorial library and recombine the mutations found at each position in the top M variants. This process is shown in Figure 2.1. For N positions, the recombinatorial library then has a maximum of MN variants and we selected the top three variants for a maximum recombinatorial library size of 81. An alternative recombination approach

is to test all possible single mutants from a given parent sequence and recombine the top three mutations at each position for a fixed recombinatorial library size of 81. However, this alternative recombination does not perform as well on the GB1 data set (Figure 3.4). Compared to these recombination strategies, the machine learning-assisted approach has the distinct advantage of providing estimates for the variability at each position (as opposed to taking the top three mutations at each).

Experimental Burden for Single-Mutation Walks

The comparison of the number of variants necessary for a single-mutation walk is a central argument of the main text and deserves extra explanation. Protein engineers often aim for 95% library coverage (9, 10), or in other words a 95% probability of seeing a particular variant in the library. Assuming equal frequency of each amino acid, this number is roughly 3-fold the library size, which is often used in practice (9). Therefore, for 19 mutations away from one of the 20 canonical amino acids, 19×3=57 variants are needed for roughly 95% coverage. The single-mutation walk to identify mutations at four positions has $4 + 3 + 2 + 1 = 10$ such libraries, for a total of 570 variants.

However, a different analysis without making these assumptions can be completed for this particular library by using expressions developed by Bosley and Ostermeier (11). From this work, the probability P_i of a particular sequence i is given below, where N is the number of tested variants and f_i is the frequency at which the sequence i is expected to be present.

$$P_i = [1 - (1 - f_i)^N] \tag{3.1}$$

Rearranged to give

$$N = \frac{\ln(1 - P_i)}{\ln(1 - f)} \tag{3.2}$$

As stated previously, a typical desired library coverage is $P_i = 0.95$ for 95% coverage, but the choice of codons can have a strong effect on the value of f_i. Assuming equal representation of the 19 codons gives $N \approx 55.4$, or 554 variants for the 10 libraries needed. However, the authors of the landscape used NNS/NNK codons, which encode for 20 amino acids with 32 codons. The least frequent amino acid encoded with these codons (methionine) occurs at a frequency of 1/32, requiring $N \approx 94.4$, or 944 variants. To balance the degenerate codon complexity and amino acid coverage, protein engineers employ the use of NDT/VHG/TGG codons, also

known as the 22c-trick [14], in which methionine occurs 1/22 times for $N \approx 64.4$, or 644 variants over 10 libraries.

From a protein engineer's perspective, a comparison to 644 variants is likely the most pertinent. However, to provide directed evolution with a stronger baseline for comparison, we have used 570 variants as a more stringent comparison with less data available, which is obtained from applying the 3-fold oversampling rule [12] to 10 libraries containing 19 desired variants. In any case, we empirically observe that the single-mutation walk performs similarly to the ML approach trained with 300–400 variants (Figure 3.3), which is significantly fewer than the number of variants shown in the main text.

Comparison between Evolutionary Approaches

To compare the distribution of fitness values of the optimal variants found by the described directed evolution methods, shallow neural networks were trained with 470 randomly selected input variants. From this, 100 predictions were tested, for a total screening burden equivalent to the single-mutation walk. In this work, the number of variants tested was determined by comparison to another method (a single-mutant walk) in the previous section and the ratio of training variants versus predicted variants was set through experimental convenience (the size of a 96-well deep-well plate). However, from a modeling perspective, these design choices could be optimized to increase the expected fitness improvement (Figure 3.3). Specifically, the number of rounds and the number of variants tested in each round Histograms of the highest fitnesses found by these approaches are shown in Figure 3.2A and reiterated as empirical cumulative distribution functions in Figure 3.2B.

As shown in Figure 3.2, with the same number of variants screened, machine learning-assisted evolution reaches the global optimum fitness value in 8.2% of 600 simulations, compared to 4.9% of all starting sequences reaching the same value through a single-mutant walk and 4.0% of simulated recombination runs. Additionally, on this landscape the machine-learning approach requires about 30% fewer variants to achieve final results similar to the single-mutant walk with this analysis. Perhaps more importantly, a single-mutant walk is much more likely to end at low fitness levels compared to approaches that sample the combinatorial library directly. To this end, the machine learning approach has an expected fitness value of 6.42, compared to 5.41 and 5.93 for the single step walk and recombination, respectively. These differences are statistically significant (p-values of 3.3×10^{-62}

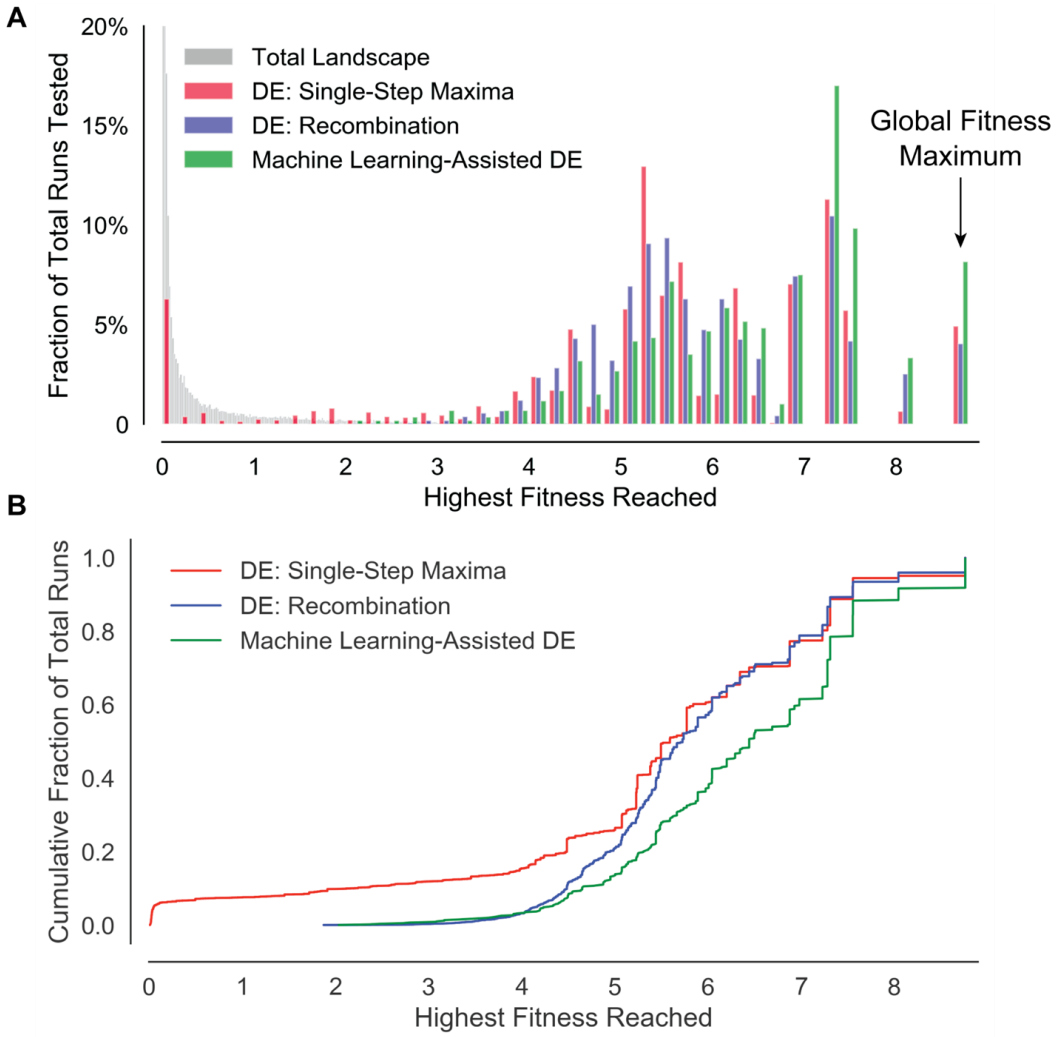


Figure 3.2: (A) Highest fitness values found by directed evolution and directed evolution assisted by machine learning. The distribution of fitness peaks found by iterative site-saturation mutagenesis from all labeled variants (149,361 out of 204 possible covering four residues) is shown in red. The distribution of fitness peaks found by 10,000 recombination runs with an average of 570 variants tested is shown in blue. The distribution of the highest fitnesses found from 600 runs of the machine learning-assisted approach is shown in green. In all approaches, 570 variants are tested. For reference, the distribution of all measured fitness values in the landscape is shown in gray. (B) The same evolutionary distributions are shown as empirical cumulative distribution functions, where the ordinate at any specified fitness value is the fraction of evolutionary runs that reach a fitness less than or equal to that specified value. Machine learning-assisted evolution walks are more likely to reach higher fitness levels compared to conventional directed evolution.

.

and 2.3×10^{-18}, respectively) as determined by a one-tailed t-test with unequal variance.

Interestingly, the accuracy of the machine learning models as determined on a test set of 1000 random variants not found in the training set can be quite low (Pearson's r = 0.41 with stdev 0.17). However, this level of accuracy as measured by Pearson's r appears to be sufficient to guide evolution. Although perfect accuracy does not seem to be necessary, if the accuracy of the trained model is so low that predictions are random guesses, this approach cannot be expected to outperform a single mutant walk (Figure 3.3). As an algorithm, evolution is focused on identifying optimal variants, and developing a measure of model accuracy biased toward correctly identifying optimal variants will likely improve model selection. This validation experiment gave us confidence that machine learning-assisted directed evolution can find improved protein variants efficiently.

3.4 Discussion

On an empirical protein fitness landscape, we have validated a machine learning-guided directed evolution approach, demonstrating its increased efficacy of both discovering the fitness maximum and in increasing the expected improvement after evolution. However, it is important to stress that this is simply one application and that this landscape was initially obtained because it was known to contain epistatic mutations [10]. This allowed us to validate our models' ability to capture epistatic landscapes, and provided a ground truth test bed for comparing engineering methods. In this proof of concept work, we heavily restrict modeling to base machine learning models available in scikit-learn [15] and have made design constraints that are heavily influenced by current workflows in protein engineering.

Nonetheless, there are many improvements that can be made when applying this method to directed evolution campaigns. While the immediate improvement that comes to mind may be to test more exotic machine learning models, every step of this process can be fine-tuned. The distribution of initial sequences that serve as the training set can be selected to be more informative. The manner in which these sequences are encoded (either through physicochemical properties [16, 17] or learned embeddings [18, 19]) can capture stronger prior information. The distribution of the number of sequences used to train and test (and the number of rounds of experimentation) can be adjusted and guided by active learning [1]. These improvements are actively being explored by our group and others.

A particularly important improvement that warrants further discussion is in the evaluation of predictive models. Much of the current literature in applying machine learning to protein engineering focuses on increased accuracy for existing data in regression tasks, and two superlative examples comparing multiple approaches may be found by Xu and Rao [17, 19]. However, it is not immediately clear that increasing predictive model accuracy in these cases will enable the *design of new* sequences with higher fitness. This remains an open question to the field [20], and current approaches are summarized in Chapter 1. Nevertheless, as machine learning is increasingly applied to protein engineering [21], it will be ever more important to understand the mechanisms that generate these data, whether driven by nature or humans.

3.5 Materials and Methods

Fitness values were provided for 149,361 out of 160,000 total possible sequences covering four positions in human protein GB1, where fitness was defined as the enrichment of folded protein bound to IgG-Fc antibody as measured by coupling mRNA display with next-generation sequencing [10]. We only use measured sequences and did not incorporate imputed values of variants that were not measured directly. Three directed evolution approaches were simulated on this landscape: A) a single mutation walk, B) simulated recombination, and C) directed evolution with machine learning.

For A), the single mutation walk, the algorithm proceeds as follows:

1. From a starting sequence, every possible single mutation (19N variants for N positions) is made and evaluated.
2. The best single mutation is fixed in the reference sequence, and the position it was found in is locked from further editing.
3. Steps 1. and 2. are repeated until every position has been tested, for a total of four rounds to cover four positions.

For B), simulated recombination proceeds by selecting 527 random variants and recombining the mutations found in the top three variants, for an average of 570 variants tested over 10,000 simulations.

For C), directed evolution with machine learning proceeds as follows:

1. 470 randomly selected sequences in the combinatorial space are used to train shallow neural networks with randomized hyperparameter search from 4-fold cross-validation based on Pearson's r. The scikit-learn implementation of multilayer perceptrons is used [15] with one-hot encodings of amino acid sequence. Errors are then calculated based on 1000 randomly selected variants that were not present in the training set.

2. The optimal model is used to predict the top 100 sequences, or roughly the screening capacity of a plate.

3. The highest true fitness value in this predicted set of 100 sequences and the training set of 470 is the maximum fitness value found.

This process was repeated with different numbers of random sequences in step 1 to simulate lower model accuracies, the results of which can be seen in Figure 3.3. In Figure 3.4, 100 variants were used as the size of the predicted library test for its similarity to the screening capacity of a 96-well plate. With 570 total variants (SI Appendix, Library Coverage), this leaves 470 variants for the input library in step I for an equal screening burden, assuming 95% coverage of 19 mutations from wild type at each position.

3.6 Supplemental Information

Additional comparison to other evolutionary techniques

Empirical Cumulative Distribution Functions (eCDFs) are shown for increasing amount of data input in Supporting Figure 3.3. In Supporting Figure 3.3, 200 simulated evolutions are tested for each of the machine learning-assisted methods, with the exception of N=470, where 600 simulations are performed, as this example is directly related to the experimental burden of the iterative single-site saturation approach.

In Supporting Figure 3.4, we compare the results of various alternative evolutionary methods that are less efficient. This is a direct extension of Figure 3.2B in the main text, but we leave the abridged version in the main text for simplicity.

In Figures 3.3 and 3.4, lines shifted toward the right are more likely to identify sequences with higher fitness. The cumulative fraction is shown on the ordinate axis, and fitness value on the abscissa. The highest fitness value from the top 100 sequences (roughly the smallest batch size, as screening is typically done in 96-well deep-well plates) from each model trained with N sequences is shown to demonstrate the effect

of increased training data. Therefore, the total screening burden for each line is $N + 100$. With 570 sequences measured (in black), the machine learning-assisted evolution approach reaches the global optimum fitness value in 8.4% of simulations, compared to 4.9% of all starting sequences (in blue). The machine learning-assisted evolution approach only requires between 300 and 400 total tested sequences to perform similarly to directed evolution (570 sequences). Therefore, the directed evolution approach requires about 42% more variants tested to achieve similar results on this landscape. However, perhaps a more important metric is the expected fitness value obtained by each method, summarized below in Table 3.1.

Table 3.1: Expected value for fitness reached and fraction of simulated evolutions that reach the maximum fitness value of various evolutionary strategies.

	Expected Fitness Reached (equivalent screening)	Fraction of Runs that reach the Maximum
ProSAR	3.00	0.20%
Recombining 3 Best Single Mutations at Each Position	4.07	1.18%
1000 Random Combinatorial Sequences	5.04	0.40%
Single Step Mutation Walk	5.41	4.91%
DE+ML (300 total sequences)	5.46	3.5%
DE+ML (400 total sequences)	5.74	2.0%
Testing random sequences, and recombining the top 3	5.93	4.03%
DE+ML (570 total sequences)	**6.42**	**8.17%**

Other controls included in Supporting Figure 3.4 are for random combinatorial sequences, from which the highest fitness from 1000 random samples is provided (in gray); two different methods of recombination (in cyan and gold); and a ProSAR-like algorithm (in red). In cyan, recombination from the top 3 single mutants at each position from a reference parent are shown. The top 3 mutants from a random combinatorial search of all positions is shown in gold (with an average of 570 sequences searched).

Our implementation of ProSAR is based on the Partial Least Squares (PLS) algorithm for a linear model for single mutations established by Fox and coworkers [22, 23]. Specifically, the PLS implementation by scikit-learn is trained with data from 569 random sequences (optimized over the number of components kept). From the PLS decomposition, the coefficients for the linear contribution from each mutation is determined, and the most positive mutation at each position is kept. We call this approach "ProSAR-like", as the exact implementation of ProSAR can be fairly subjective. See Supporting Information (Detailed description of a round): in *Improving catalytic function by ProSAR-driven enzyme evolution* by [23].

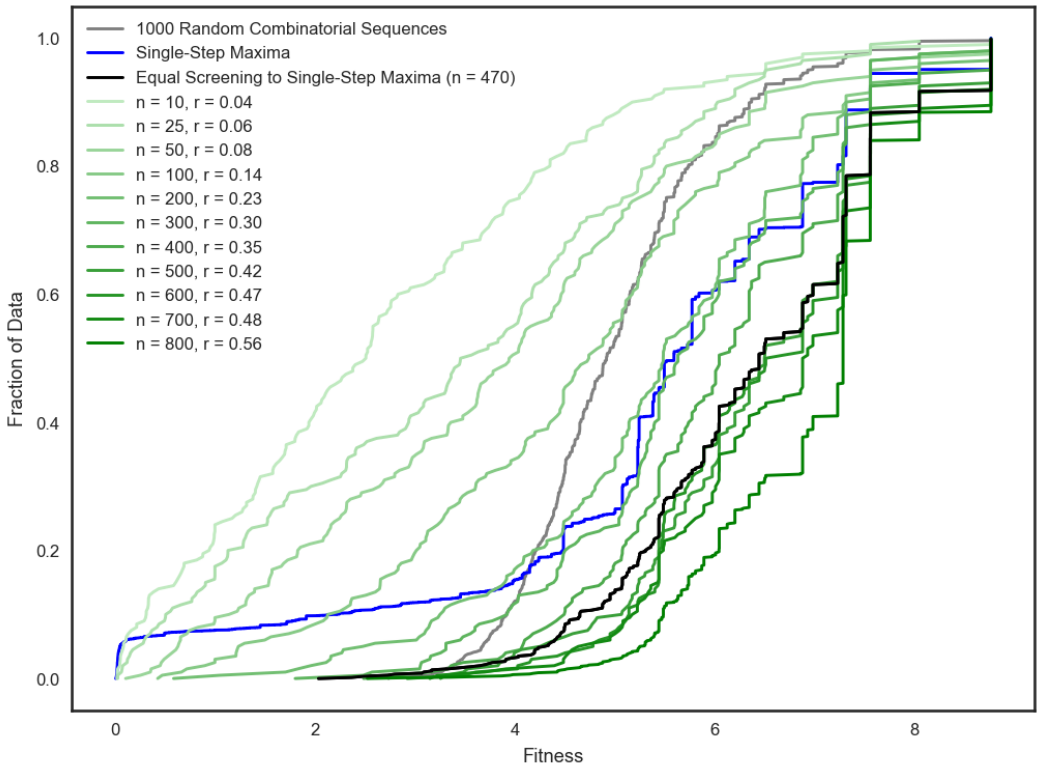


Figure 3.3: Highest fitnesses found with less accurate models.

The low performance of ProSAR on this landscape is worth discussing. ProSAR was developed to analyze previously identified mutations at different positions, such that each position typically only has one (maybe two) mutations to consider. A base model with linear contributions at these positions supported their evolution. However, in our recombination landscape of a small number of positions with known epistasis (nonlinear effects), thus ProSAR should not be expected to find optimal solutions (and does not outperform other methods tested).

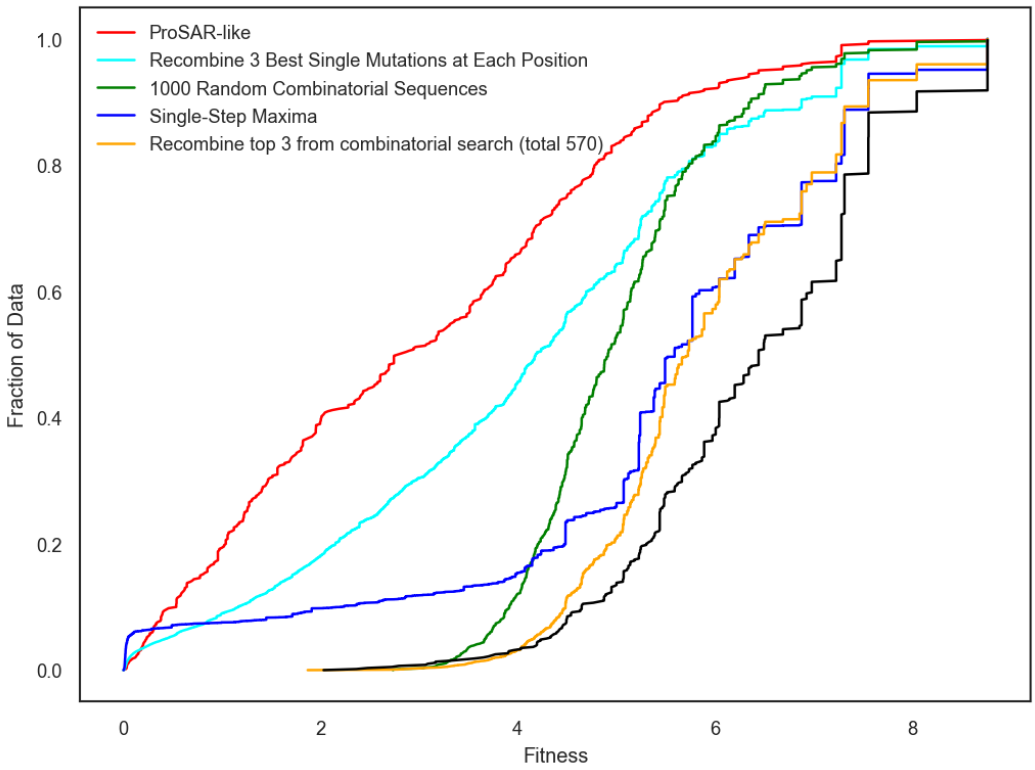


Figure 3.4: Highest fitnesses found with other directed evolution approaches.

3.7 Bibliography

References

1. Romero, P. A. & Arnold, F. H. Exploring protein fitness landscapes by directed evolution. *Nature Reviews Molecular Cell Biology* **10,** 866–876. doi:10.1038/nrm2805 (2009).

2. Cheng, F., Zhu, L. & Schwaneberg, U. Directed evolution 2.0: improving and deciphering enzyme properties. *Chemical Communications* **51,** 9760–9772 (2015).

3. Schuster, S. C. Next-generation sequencing transforms today's biology. *Nature Methods* **5,** 16–18 (2008).

4. Goldberg, M. BioFab: applying Moore's law to DNA synthesis. *Industrial Biotechnology* **9,** 10–12 (2013).

5. Diefenbach, X. W. *et al.* Enabling biocatalysis by high-throughput protein engineering using droplet microfluidics coupled to mass spectrometry. *ACS Omega* **3,** 1498–1508 (2018).

6. Packer, M. S. & Liu, D. R. Methods for the directed evolution of proteins. *Nature Reviews Genetics* **16,** 379–394 (2015).

7. Smith, J. M. Natural selection and the concept of a protein space. *Nature* **225,** 563–564 (1970).

8. Bershtein, S., Segal, M., Bekerman, R., Tokuriki, N. & Tawfik, D. S. Robustness–epistasis link shapes the fitness landscape of a randomly drifting protein. *Nature* **444,** 929–932 (2006).

9. Starr, T. N. & Thornton, J. W. Epistasis in protein evolution. *Protein Science* **25,** 1204–1218 (2016).

10. Wu, N. C., Dai, L., Olson, C. A., Lloyd-Smith, J. O. & Sun, R. Adaptation in protein fitness landscapes is facilitated by indirect paths. *Elife* **5,** e16965. doi:10.7554/eLife.16965 (2016).

11. Fowler, D. M. & Fields, S. Deep mutational scanning: a new style of protein science. *Nature Methods* **11,** 801. doi:10.1038/nmeth.3027 (2014).

12. Reetz, M. T., Kahakeaw, D. & Lohmer, R. Addressing the numbers problem in directed evolution. *ChemBioChem* **9,** 1797–1804 (2008).

13. Bosley, A. D. & Ostermeier, M. Mathematical expressions useful in the construction, description and evaluation of protein libraries. *Biomolecular Engineering* **22,** 57–61 (2005).

14. Kille, S. *et al.* Reducing codon redundancy and screening effort of combinatorial protein libraries created by saturation mutagenesis. *ACS Synthetic Biology* **2,** 83–92 (2013).

15. Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830 (2011).

16. Cadet, F. *et al.* A machine learning approach for reliable prediction of amino acid interactions and its application in the directed evolution of enantioselective enzymes. *Scientific Reports* **8,** 1–15 (2018).

17. Xu, Y. *et al.* A Deep Dive into Machine Learning Models for Protein Engineering. *Journal of Chemical Information and Modeling* (2020).

18. Yang, K. K., Wu, Z., Bedbrook, C. N. & Arnold, F. H. Learned protein embeddings for machine learning. *Bioinformatics* **34,** 2642–2648. doi:10.1093/bioinformatics/bty178 (2018).

19. Rao, R. *et al. Evaluating protein transfer learning with TAPE* in *Advances in Neural Information Processing Systems* (2019), 9686–9698.

20. Brookes, D. H. & Listgarten, J. Design by adaptive sampling. *arXiv* (2018).

21. Yang, K. K., Wu, Z. & Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nature Methods* **16,** 687–694. doi:10.1038/s41592-019-0496-6 (2019).

22. Fox, R. J. *et al.* Optimizing the search algorithm for protein engineering by directed evolution. *Protein Engineering* **16,** 589–597 (2003).

23. Fox, R. J. *et al.* Improving catalytic function by ProSAR-driven enzyme evolution. *Nature Biotechnology* **25,** 338 (2007).

Chapter 4

SIGNAL PEPTIDES GENERATED BY ATTENTION-BASED NEURAL NETWORKS

1. Wu, Z. *et al.* Signal Peptides Generated by Attention-Based Neural Networks. *ACS Synthetic Biology* **9,** 2154–2161 (2020).

Contributions Statement: Z.W. conceived and directed this study. K.K.Y., A.L., and Z.W. obtained training data and trained the models. Z.W., M.J.L., and D. Wernick planned the *in vivo* experimental validation. M.J.L., and A.B. performed the experimental validation. Z.W. analyzed the experimental results.

4.1 Abstact

Short (15-30 residue) chains of amino acids at the amino termini of expressed proteins known as signal peptides (SPs) specify secretion in living cells. We trained an attention-based neural network, the Transformer model, on data from all available organisms in Swiss-Prot to generate SP sequences. Experimental testing demonstrates that the model-generated SPs are functional: when appended to enzymes expressed in an industrial *Bacillus subtilis* strain, the SPs lead to secreted activity that is competitive with industrially used SPs. Additionally, the model-generated SPs are diverse in sequence, sharing as little as 58% sequence identity to the closest known native signal peptide and 73% ± 9% on average.

4.2 Introduction

For cells to function, proteins must be targeted to their proper locations. Over one-third of the bacterial proteome that is synthesized in the cytoplasm is exported outside of it, and as a core requirement, the pathways that control localization are highly conserved across all domains of life [1]. To direct a protein through secretion pathways, organisms encode instructions in a leading short peptide sequence (typically 15-30 amino acids) called a signal peptide (SP) [2]. SPs direct peptide chains to various export pathways, including the well-characterized Sec-[3–5] and Tat-mediated pathways [6, 7].

SPs have been engineered for a variety of industrial and therapeutic purposes, including increased export for recombinant protein production [2, 8] and increasing the

therapeutic levels of proteins secreted from industrial production hosts [9]. Secretion facilitates protein production by removing stress caused by protein accumulation in the cytoplasm, as well as by placing the burden of separation on the cells, which simplifies downstream processing [10].

Due to the utility and ubiquity of protein secretion pathways, a significant amount of work has been invested in identifying SPs in natural protein sequences. Much of this work was pioneered by the groups behind the SignalP web server, which first used artificial neural networks [11] and hidden Markov models [12] and now leverages modern deep learning architectures to model SPs [13]. An additional tool from the SignalP team, TargetP, is capable of identifying SP sequences and classifying them by the pathway used and the targeted intracellular or extracellular location [13].

While this is a significant step toward modeling SP sequences from proteomic data, the challenging task of generating a SP sequence has yet to be validated *in vivo*. Indeed, the task of generating protein sequences of any kind is just beginning to be tackled [14–18]. Given a desired protein to target for secretion, there is no universally-optimal directing SP [19, 20] and there is no reliable method for generating a SP with measurable activity. Instead, libraries of naturally-occurring SP sequences from the host organism or phylogenetically-related organisms are tested for each new protein secretion target [20, 21]. That these libraries give functional SPs for new proteins is due to inherent "transferability" of SP sequences among multiple targets: empirically, roughly 50% [20] to 68% [19] of natural SPs paired to a protein secretion target produce measurable activity.

Although at one time the space of functional SP sequences was hypothesized to be quite large under the helical hairpin hypothesis [22], subsequent research found that nature has designed SP sequences to interact with the necessary translocons in various pathways [23]. While researchers have attempted to generalize our understanding of SP–protein pairs by developing general SP design guidelines, those guidelines are heuristics at best and are limited to modifying existing SPs, not designing new ones [2, 24, 25].

Here we present a machine translation model for generating SP sequences that have a high probability of being functional. Specifically, we trained a Transformer model [26] to predict SPs given the mature protein sequences of proteins annotated with SPs in Swiss-Prot [27] from all available organisms. These generated sequences are predicted by SignalP to have high probability of functioning as SPs. Upon *in vivo* validation in a gram-positive production organism, we find that 48% of

constructs with generated SPs lead to secreted enzyme activity comparable to SPs used industrially. The functional generated sequences share as little as 58% sequence identity to the closest natural SP.

4.3 Results

Model Description

We cast the SP generation problem as a translation problem by using the mature protein with the SP sequence removed as the source and the corresponding SP sequence as the output sequence. We employ the Transformer encoder-decoder architecture as first described by Vaswani *et al*. [26] that leverages an attention mechanism [28] which weights different positions over the entire sequence in order to determine a representation of that sequence, and remains a state-of-the-art architecture for machine translation between human languages [29, 30]. Recent work has also applied the Transformer model to extract information from protein sequences for use in downstream protein function prediction and engineering tasks [31, 32].

Training Objective

We apply the Transformer architecture to SP prediction by treating each of the amino acids as a token (cf. machine translation, where words, characters, or subwords are tokens). The Transformer encoder maps an input sequence of tokens (the protein amino acids) to a sequence of continuous representations. Given these representations, the decoder then generates an output sequence (the SP amino acids) one token at a time. Each step in this generation depends on the generated sequence elements preceding the current step and continues until a special <END OF SP> token is generated. Figure 4.1 illustrates the modeling scheme. During training, we pass the decoder the true target SP. Training details can be found in Methods.

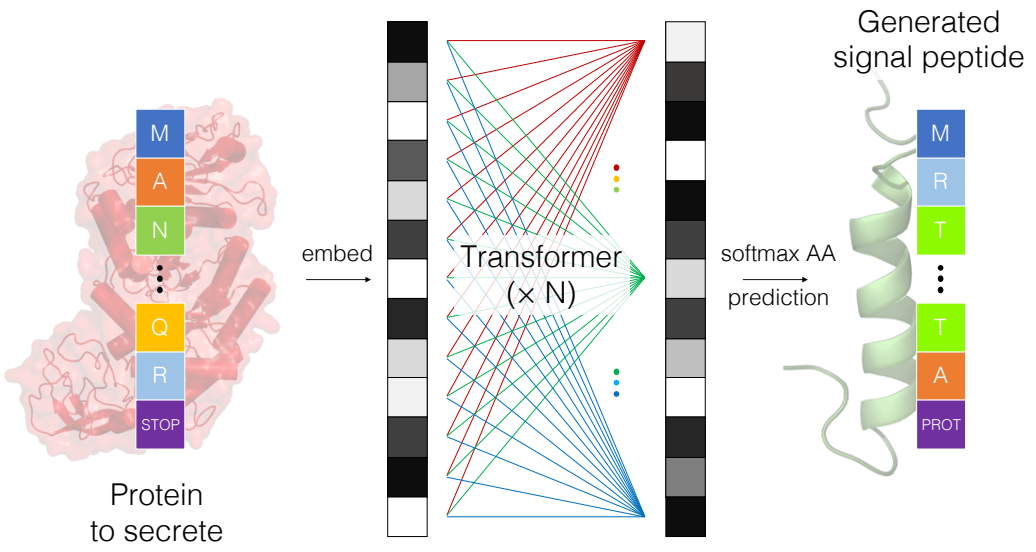


Figure 4.1: Sequence-to-sequence modeling for signal peptide (SP) amino acid sequences. During training, the first 100 amino acids of the protein are tokenized and embedded.

.

Training Data

From Swiss-Prot [27] we were able to extract over 40,000 SP–protein pairs from all domains of life. Protein secretory pathways are highly conserved, and others have found that incorporating data from all available organisms boosts accuracy in secretion prediction [13]. Additionally, we elect to train with SP-protein sequences over SP sequences alone, as experimental evidence suggests strong dependence on the protein sequence [2, 13, 19]. We selected sequence length maximum cutoffs of 70 amino acids and 105 amino acids for the SP and protein, respectively, to capture important motifs while keeping sequences short for more efficient training.

Model inputs are one-hot encodings of amino acid sequences of proteins to be secreted without their corresponding signal peptide. Mature proteins were padded or truncated to 105 residues, by observing that the loss during training did not decrease with longer input protein sequences, and the memory and computation required by the Transformer architecture scales quadratically with the sequence length. After truncation and removing duplicates, 25,000 SP–protein pairs remained, which were split randomly into training (80%) and validation (20%) sets. While we chose to restrict our search to the reviewed portion of UniProt (Swiss-Prot), most of the SPs returned were identified by computational annotation, and a future alternative is to

incorporate sequences identified in TrEMBL, allowing a larger training set for better model prediction. Model outputs are one-hot encodings of signal peptide amino acid sequences, which are padded or truncated to 70 residues.

In addition to training on the full dataset, we also trained on filtered subsets of the full dataset for which we removed sequences with ≥ 75%, 90%, 95%, or 99% sequence identity to 28 enzymes from 4 families we selected for experimental validation (further described below) in order to test the model's ability to generalize to distant protein sequences. The Transformer model was then trained on each of these filtered datasets and used to generate sequences.

Machine Sequence Generation

Given a trained model that predicts sequence probabilities, there are many methods by which protein sequences can be generated [14, 16]. One such method is beam search [33], which generates a sequence by taking the most probable amino acid additions from the N-terminus. In traditional beam search, the size of the beam refers to the number of unique hypotheses with highest predicted probability for a specific input that are tracked at each generation step. For example, a beam size of 5 generates hypotheses from the N to C terminus, keeping the 5 most probable sequences as the sequence grows. In this study, we attempt to generate "generalist" SPs, which have higher probability of functioning across multiple input protein sequences. To this end, we employed an alternate form of beam search, which we call "mixed input beam search" with a beam size of 5 over the decoder in identifying SPs. Our mixed input beam search generates SP hypotheses for multiple protein inputs, keeping the SP sequences with highest predicted probabilities. This generation process reflects the natural SPs' transferability between proteins to secrete, as 50-68% of natural SPs from related species exhibit measurable function when tested against specific enzymes [19, 20]. By providing the Transformer model with multiple enzymes, the model has an opportunity to generate a sequence with high likelihood given multiple inputs, rather than being forced to generate a SP for an input it is unsure about.

For this study, we aimed to identify novel SPs (new amino acid sequences) and test them for secretion of ten enzymes across four families (amylases, lipases, proteases, and xylanases) in an industrial gram-positive bacterial (*Bacillus subtilis*) host. In addition to the ten enzymes tested, we also provided 31 other enzymes as inputs to generate SPs, in an effort to increase the transferability of generated SPs to multiple enzymes. The enzymes and the SPs generated for them can be found in

Supplementary Section VII. Predictions were made based on models trained on the four cutoffs for sequence identity described above. The generated SP sequences from each cutoff (4 SPs for each target enzyme) were appended to different protein target sequences to test with SignalP. The generated SPs also showed high probability of functioning as predicted by SignalP 5.0 (average probability 90.4% ± 17.1%, Supplemental Section II) and also contain many of the motifs common to SPs (positively charged N-terminus, hydrophobic core, and terminal AXA motif). While these heuristics could also be used to generated SPs, we find that the machine-learning approach generates SPs with significantly higher predicted probability of functioning than those generated by heuristics (p-value = 5×10^{-28}; Supplemental Section II), which agrees with reported experimental difficulty in applying heuristics to designing SPs [23]. We also provide comparisons to sequences generated by HMMER [34] and a variational autoencoder [35] in Supplemental Section II.

Secreted Enzyme Activity Validation

We then tested the predicted SPs by expressing SP-protein pairs in a *Bacillus subtilis* host strain used for secretion of industrial enzymes. We expressed ten enzymes: 5 amylases, 1 lipase, 2 proteases, and 2 xylanases. Functional secretion was determined by testing fermentation supernatants with the corresponding enzyme activity assay, as described in Supplemental Table 4.3.

For the ten enzymes, we tested 1) SPs generated by the Transformer model 2) industrial SPs native to *Bacillus subtilis* (positive controls) and 3) SPs generated using random source amino acid sequences¬¬¬. The sequences for 1) and 2) can be found in Supplementary File 1. For the positive controls, we used six SPs represented in previous studies for industrial levels of protein secretion (AprE, LipB, YbdG, YcnJ, YkvV, and YvcE) [19, 20, 36]. For 3), output SPs were generated by the Transformer model for input protein sequences, which were made by drawing randomly from random amino acid distributions following a) the *Bacillus* amino acid distribution, b) the bacterial amino acid distribution, and c) a uniform amino acid distribution. The sequences can be found in Supplemental Section I, and the functional classification results are summarized in Table 4.1. The measured enzyme activities for each construct, as well as details for their functional classification, can be found in Supplemental Section III. A total of 163 unique constructs were tested.

Functional classification is summarized in Supplemental Section III, where enzyme activity in the supernatant is plotted for visual comparison. SPs Generated for

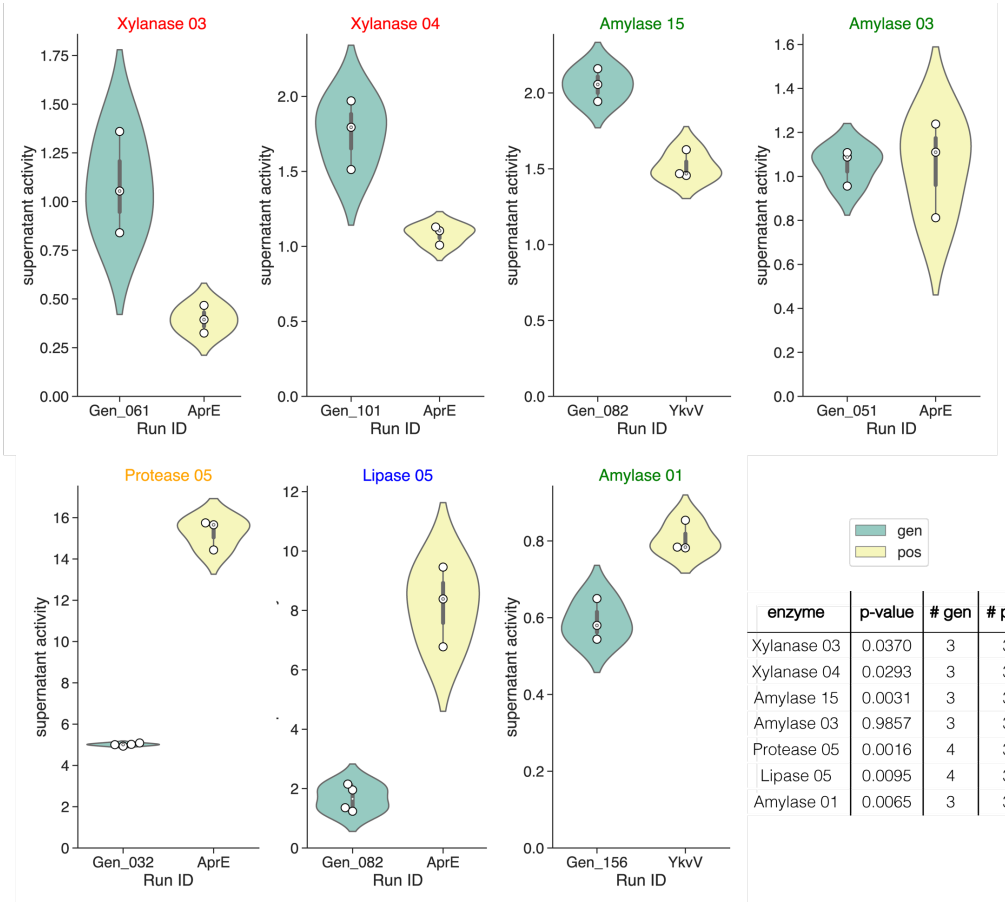
Table 4.1: Summary of protein-SP constructs that are functional.

	Num Functional	Num Tested	Percent Functional
SPs Generated for Random Inputs	1	18	6%
Natural SPs	27	34	79%
Generated SPs	53	111	48%

Random Inputs were generated by the Transformer model given randomized amino acid sequences for the target protein, as detailed in Methods. Positive controls are naturally occurring *Bacillus* signal peptides. Generated SPs were generated for 41 proteins through mixed input beam search.

Using native SPs, 79% of constructs with 6 commonly used SPs resulted in secreted activity. We were pleased to find that a substantial fraction (48%) of the constructs containing a generated SP also resulted in significant secreted activity. SPs generated for random protein inputs were much less likely to lead to secreted activity. Only one construct containing an SP generated given random amino acid sequences gave some supernatant activity (Protease 05, Supplemental Section III), which indicates that, in general, a real protein sequence is required for generating a sensible SP and the model is not relying on other artefacts for generation. Additionally, for the 21 generated functional SPs that were tested with multiple proteins, all 21 were functional for all proteins with which they were tested.

Generated SP-Enzyme Constructs Exhibit Activity Comparable to Natural Constructs. The model-generated SP-enzyme constructs are not only functional; they also exhibit activity similar to that of constructs with natural SPs. This is shown in Figure 4.2, which illustrates the highest performing natural and generated SP for each enzyme tested with both. Activities for all generated constructs can be found for comparison in Supplemental Section IV. Of the tested enzymes, approximately half exhibited higher or comparable secreted activity with machine-generated SPs. Thus, the generated SPs offer comparable and sometimes significantly higher activity compared to natural SPs, even with a generative model that was not specifically trained to optimize secretion levels.



enzyme	p-value	# gen	# pos
Xylanase 03	0.0370	3	3
Xylanase 04	0.0293	3	3
Amylase 15	0.0031	3	3
Amylase 03	0.9857	3	3
Protease 05	0.0016	4	3
Lipase 05	0.0095	4	3
Amylase 01	0.0065	3	3

Figure 4.2: Generated signal peptides enable secreted enzyme activities that are comparable to natural SPs. The highest-performing natural (labeled "pos") and machine-generated (labeled "gen") SPs are shown for the 7 enzymes where both were tested. Of these 7 enzymes, 4 exhibited the same or higher supernatant activity with generated SPs (top row), and 3 exhibited higher supernatant activity with native *Bacillus* SPs (bottom row). P-values are provided for reference for a two-sided t-test with unequal variance for two independent samples of scores, where the null hypothesis is that the samples have identical expected values.

Generated Constructs Are Diverse in Sequence

The generated SPs occupy regions of sequence space that are not known to have been explored by naturally occurring SPs. The input protein sequences were removed at various sequence identity cutoffs from the training set to ensure that predictions were made for enzyme sequences that the trained model had never seen before. However, we did not specifically select for SP sequences that met a specified diversity threshold, as can be done to ensure sequence diversity [37]. Interestingly, functional generated SPs share on average 73% ± 9% and as little as 58% sequence identity to the closest SP in Swiss-Prot (Figure 4.3A). For comparison, we also withheld 256 random natural SPs (82% ± 10%) and found that the functional generated SPs are significantly more diverse than these natural SPs (p-value = 3.3×10^{-7}). Multiple sequence alignments (MSAs) for each of the best generated SPs identified for each enzyme can be found in Supplemental Section VI. We show the MSA for the most distant sequence in Figure 4.3B. In general, the generated SP retains characteristics of other natural SPs, such as a positively charged N-terminus, hydrophobic core, and AXA motif, while sharing low sequence identity (as low as 58%).

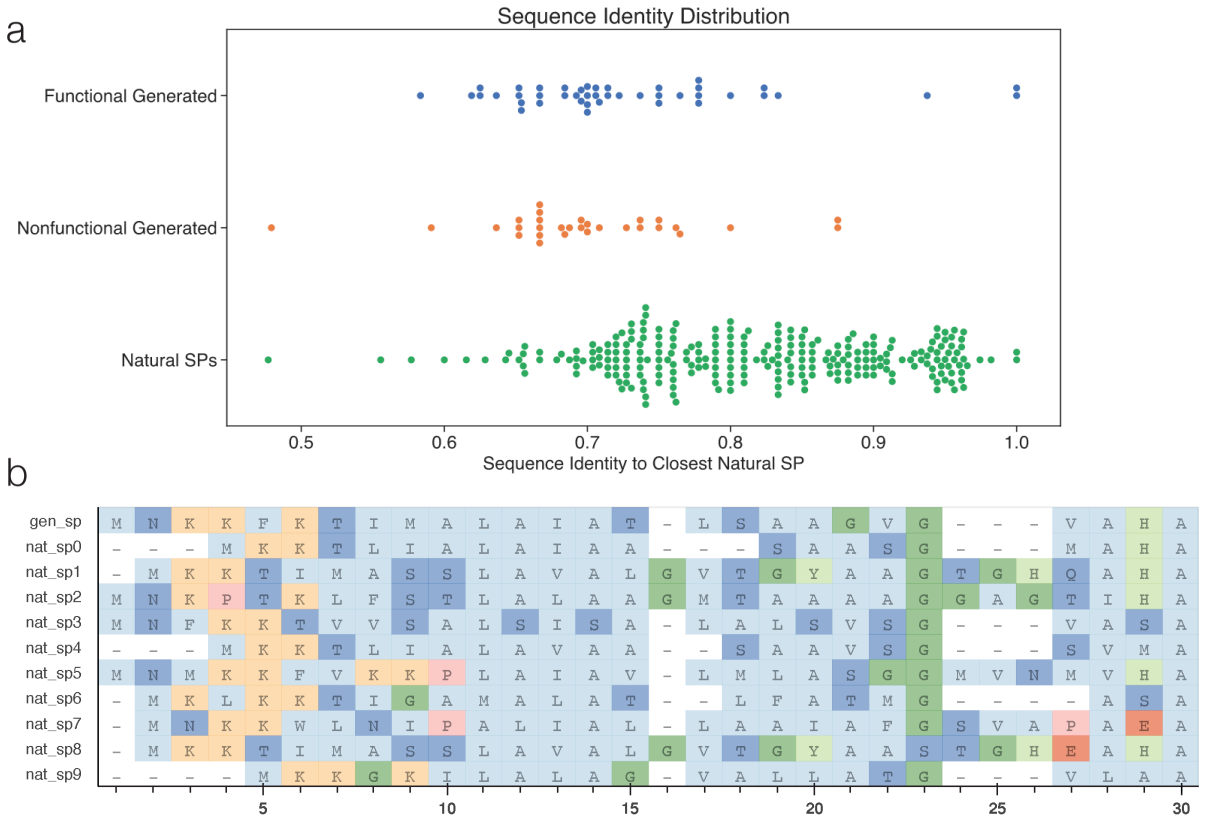


Figure 4.3: a) Percent sequence identity of various SPs to the closest matching natural SPs in Swiss-Prot, including 1) Functional Generated SPs (73% ± 9%), 2) Nonfunctional Generated SPs (70% ± 8%), and 3) a withheld set of 256 Natural SPs (82% ± 10%). Functional generated SPs have statistically significant lower percent identity (p-value = 3.3×10^{-7}). b) Multiple sequence alignment of the most diverse functional generated SP (58% identity to closest natural SP) with native SPs. Color groups follow those in ClustalW [38].

SignalP Does Not Discriminate Between Functional and Non-Functional Constructs with Generated Signal Peptides

Interestingly, the functional classification accuracy of the best server, SignalP 5.0 [13] on the generated SPs is quite low. Figure 4.4 shows a receiver-operating curve (ROC) that displays true positive rate versus false positive rate for secretion probabilities generated by SignalP. As a reminder, random guessing gives an Area Under the Curve (AUC) of 0.50. SignalP performs quite poorly, with an AUC of only 0.59. However, there are a few differences in our modeling and validation approaches worth noting. First, our model is based on the Transformer architecture, whereas SignalP relies on bidirectional long short-term memory (LSTM) cells for longer range sequence interactions. Empirically, attention-based models currently have generally higher accuracy than LSTMs for protein tasks [31]. Additionally, our specific validation task of secreting functional enzyme in *Bacillus subtilis* differs from that of SignalP, which aims to assign a probability for sequences functioning as SPs from genomic data across many domains of life. Therefore, although SignalP may have the ability to discern natural SPs from other sequences, it does not appear to classify machine-generated SPs in *Bacillus* well, as previously shown by Brockmeier and coworkers [19]. This low accuracy may result from an inability to predict expression in the desired host, which SignalP is not trained for. In the future, SignalP may be adapted for specific production organisms in a feedback loop with our model, which is capable of generating functional sequences to test. We attempted to identify general protein properties from Biopython [39] that differed between the functional and nonfunctional SPs, but were unable to identify any statistically significant differences (Supplemental Section V).

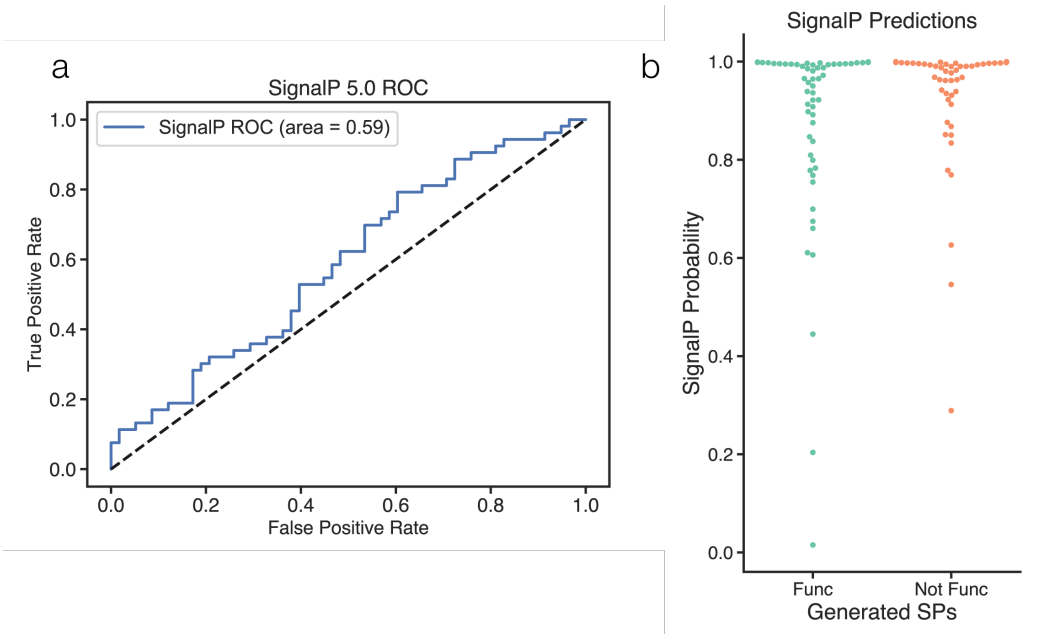


Figure 4.4: a) Receiver Operating Characteristic (ROC) curve for the prediction of functional constructs with machine-generated SPs. The SignalP 5.0 web server, an exemplary tool for natural SP annotation, performs poorly on this task, with AUC=0.59 (compared to 0.50 for random guess). b) Probability predictions for functional and nonfunctional generated SP constructs. Most constructs are predicted to be functional with high probability.

4.4 Discussion

We describe the application of a sequence-to-sequence model to generate functional peptide sequences that have not been identified in nature. These sequences accomplish the same function of directing enzyme secretion to the*B. subtilis* supernatant, yet they share as little as 58% sequence identity, and on average 73% ± 9%, to the closest-aligned recorded SP and thus explore new regions of sequence space. Enzymes with machine-generated SPs are expressed with activity levels comparable to those of natural SPs used in industrial enzyme production, despite not having been explicitly designed to maximize secretion levels.

This work builds upon existing efforts in protein sequence generation with deep learning by providing *in vivo* validation of predictions. In one other case in which predictions were validated experimentally, 24% of the malate dehydrogenase enzymes from a generative adversarial network (GAN) by in-sample generation were functional [17]. Our approach uses a sequence-to-sequence approach trained with SP-protein pairs. We were pleased to find that a high fraction of generated constructs (98%) were predicted to be functional by SignalP (Supplemental Section II), and a significant

fraction (48%) of constructs were in fact functional *in vivo*. While SignalP is optimistic in its predictions, the lower fraction that is functional *in vivo* and the volume of heuristics developed for modifying existing SPs [2, 24, 25] suggest this remains a challenging engineering task.

Interestingly, the leading existing model trained for identifying SPs are not able to accurately distinguish functional machine-generated SP sequences from those that are not functional. Because our model generates sequences that an advanced critic (SignalP) is not able to discriminate among, coordinating these two systems in an adversarial approach could increase accuracy for both sequence generation and discrimination.

Important for both natural and synthetic SPs is whether they are transferable between secretion targets and host organisms. In this study, limited to a single round of experimentation, we used a generation strategy with multiple protein inputs with the goal of maximizing the probability of the SP functioning for any protein sequence. Of the functional generated SPs tested with multiple proteins, all 21 were classified as functional when prepended to all tested proteins. With knowledge that the Transformer model can generate functional protein sequences, probing the accuracy with which this translation strategy is able to generate SPs specific to desired secretion proteins and whether these specific SPs are transferable are potential future directions. Additionally, augmenting the generation process by conditioning on desired metadata, such as the host species, as outlined recently by Madani and coworkers [16] may allow tuning SP sequences to different production organisms.

As the protein modeling field moves toward machine generation of protein sequences, our understanding of protein similarity must evolve as well. Similarities have historically been measured by a weighted alignment of linear sequences. However, we are finding that machine learning is capable of interpolating in modeled latent space to reach regions of sequence space that nature has yet to explore, as nature has significant physical limitations on its engineering strategies (and our databases, although large, are woefully incomplete). By challenging and supplementing nature's generation strategy with machine-generated sequences to more fully sample sequence space, we can unlock sequences with new properties and functions.

4.5 Materials and Methods

Model training. We trained a Transformer Encoder-Decoder with 5 layers and a hidden dimension of 550. Each layer had 6 attention heads. The model was trained

for 100 epochs with a dropout rate of 0.1 in each attention head and after each position-wise feed-forward layer. Following the original Transformer paper, we used periodic positional encodings and the Adam optimizer. We increased the learning linearly for the first 12500 batches from 0 to 1e-4 and then decayed by $n\_steps^{-0.03}$ after the linear warmup. Models were trained on 1 NVIDIA V100 GPU through a generous grant from the Caltech Amazon Web Services Compute program.

Data augmentation. We used varying sub-sequences of the mature protein sequences as source sequences in order to augment our training dataset, to diminish the effect of choosing one specific length cutoff, and to make the model more robust. For mature proteins of length L < 105, the model receives the first L – 10, L – 5, and L residues as training inputs. For mature proteins of L >= 105, the model receives the first 95, 100, and 105 amino residues as training inputs. Data for signal peptides were collected from UniProt.

Bacterial strains, DNA design, and library construction. The expression vector was constructed from the *Bacillus subtilis* shuttle vector pHT01 by removal of the BsaI restriction sites and replacing the inducible Pgrac promotor with the constitutive promotor Pveg. However, IPTG was included during expression to ensure no residual or off-site inhibition from the LacI fragment still included on the pHT vector. Signal peptide sequences predicted from the model were reverse translated into DNA sequences for synthesis using JCat [40] for codon optimization with *Bacillus subtilis* (strain 168). Each gene of interest was modeled at four homology cut-offs resulting in 4 predicted signal peptides. These 4 signal peptides were synthesized as a single DNA fragment with spacers including the BsaI restriction sites. 8 individual colonies were picked from each group of 4 predicted signal peptides. Protein sequences were selected from literature reports of enzymes expressed in *Bacillus* host systems. Supplemental Section VII lists the enzymes used in this work and their reported amino acid sequence. Signal peptide and protein DNA sequences were ordered from Twist Biosciences and cloned into their E. coli cloning vector. *Bacillus subtilis* PY97 was the base strain used for the expression of enzymes. Native enzymes that could interfere with measurement were knocked out as indicated in Supplemental Table 4.4.

The expression vector backbone, gene of interest, and SP fragments were amplified via PCR with primers including BsaI sites and assembled through Golden Gate Assembly, with a linker GGGGCT sequence (encoding Glycine and Alanine) between the generated SP and the target protein. Primers used to amplify each fragment

are listed in Supplemental Table 4.2. Each linear DNA fragment was agarose gel purified for use in Golden Gate assembly reactions. The Golden Gate reactions were performed with 700ng vector PCR product, 100ng signal peptide group PCR product, and 300 ng gene of interest PCR product in 20µl reactions (2µl 10x T4 Ligase Buffer, 2µl 10x BSA, 0.8µl BsaI-HFv2, 1µl T4 Ligase). The reactions were cycled 35 times (10min, 37°C; 5 min, 16°C) then heat inactivated (5 min, 50°C; 5min, 80°C) before being stored at 4°C for use directly.

Enzyme expression and functional characterization. All *Bacillus* strains were transformed by natural competency as previously described [41]. Transformations were plated on LB agar (10 g/l tryptone, 5 g/l yeast extract, 10 g/l NaCl, 15g/l agar) supplemented with 5µg/ml chloramphenicol and grown overnight at 37°C. Single colonies were picked and grown overnight in 96-well plates (Whatman #7701-5200) with LB containing 17 µg/ml chloramphenicol then stored as glycerol stocks. For enzyme expression, cultures were seeded from glycerol stocks into 100 µl LB media and grown overnight at 37 °C. A 10 µl aliquot of the overnight culture was transferred into 500 µl of 2xYT media (16 g/l Tryptone, 10 g/l yeast extract, 5 g/l NaCl) containing 1 mM IPTG and incubated for 48 hrs at either 30 °C, or 37 °Cwith shaking (900 rpm, 3 mm throw). Culture supernatants were clarified by centrifugation (4000 rpm, 10 min) and used directly in enzyme activity assays. Strains were grown and expressed in at least three biological replicates from each original picked colony.

Enzyme expression quantification was attempted via SDS-PAGE (BioRad Criterion 10-20 % Tris-HCl) but the observed expression level was below a quantifiable limit. Enzyme expression was too low to reliably quantify with SDS-PAGE, so the relative expression of each enzyme was approximated by activity measurements. Enzyme activity was measured in the linear response range for each substrate and reaction condition as listed in Supplemental Table 4.3. Intracellular enzyme expression was assessed by washing the cell pellet after the supernatant was removed, and then resuspending in 500 µl of 50 mM HEPES buffer with 2 mg/ml Lysozyme and incubated for 30 minutes at 37 °C. The resuspended material was centrifuged again and used directly in enzyme activity assays.

SPs Generated for Random Inputs. SPs were generated by the trained Transformer model with 99% sequence identity cutoff for randomized protein inputs following a) the *Bacillus* amino acid distribution, b) the bacterial amino acid distribution, and c) a uniform amino acid distribution. The same mixed input beam search generation approach was used as detailed in Machine Sequence Generation. These sequences

can be found below in Supplemental Section I.

4.6 Supplemental Information

Supporting Tables

Table 4.2: Primers used to generate linear DNA fragments.

Primer	Sequence
pHT vector fwd	CAGACTTTCTAGAGGTCTCATAGCGCAGCC
pHT vector rev	TGAGCTCCTCGAGGGTCTCTATATAGAGTC
Signal Peptide fwd	TCCGCCTGACCTCCATGGGGTCTCAATATG
Signal peptide rev	CTCCACCTAGCCTGATATCGGTCTCTAGCC
Gene of Interest fwd	ATTCCGCCTGACCTGGTCTCAGGCTG
Gene of Interest rev	CTGAGCCTCCACCTAGCCTGGTCTCTGCTA

Table 4.3: Enzymatic reaction conditions, where CAPS is N – cyclohexyl – 3-aminopropane – sulfonic acid

Enzyme Class	Substrate	Buffer	Reaction Conditions
Amylase	Red Starch (Megazyme)	50 mM NaCl, 50 mM BTP, 50mM citric acid, 50 mM CAPS, pH 7.0	20 µl of the cell supernatant were added to 10 µl of 2% (w/v) Red Starch substrate in buffer and the mixture was incubated at 40 °Cfor 30 min. Reactions were quenched with 50 µl of 95% ethanol and the assay plate was centrifuged at 4000 rpm for 10 min. The absorbance was measured in 60 µl of the supernatant solution at 510 nm.
Lipase	C8-PNP (4-Nitrophenyl octanoate, Sigma)	100 mM HEPES, pH 7.5, 100 mM NaCl, 20 mM $CaCl_2$, 0.1% Triton-100	10 µl of the cell supernatant were added to 90 µl of 0.8 mM C8-PNP substrate in buffer. The absorbance was measured at 405 nm kinetically for 15 min at a 31 sec interval. The initial reaction rate was calculated from the linear portion of the kinetic read.
Protease	AAPF (N-Succinyl-Ala-Ala-Pro-Phe p-nitroanilide, Sigma)	100 mM HEPES buffer, pH 7.5	10 µl of the cell supernatant were added to 90 µl of 2mM AAPF substrate in buffer. The absorbance was measured at 410 nm kinetically for 15 min at a 31 sec interval. The initial reaction rate was calculated from the linear portion of the kinetic read.
Xylanase	Azo-Xylan Birchwood (Megazyme)	50mM NaCl, 50mM Bis-tris propane (BTP), 50mM Citric acid, 50 mM CAPS, pH 7.0	20 µl of the cell supernatant were added to 20 µl of 2% (w/v) Azo-Xylan substrate in buffer and was incubated at 40 °Cfor 30 min. Reactions were quenched with 100 µl of 95% Ethanol and the assay plate was centrifuged at 4000 rpm for 10 min. The absorbance was measured in 90 µl of the supernatant solution at 590 nm.

Table 4.4: Enzyme production strains.

Strain	Modifications	Reference
Bacillus subtilis PY79 base strain	ΔnprE ΔaprE Δepr Δmpr ΔnprB Δvpr Δbpr ΔsigF ΔskfA Δxpf ΔlytC ΔsdpC	Provided by BASF
BS – Prot	PY79 base strain ΔskfA ΔspbC ΔskfA	Provided by BASF
BS – Amy	PY79 base strain ΔskfA ΔspbC ΔskfA ΔAmyE	Provided by BASF
BS – Xyl	PY79 base strain ΔskfA ΔspbC ΔskfA ΔxynA ΔxynC	Provided by BASF
BS – Lip	PY79 base strain ΔskfA ΔspbC ΔskfA ΔestA ΔestB	Provided by BASF

Table 4.5: Distribution of Protein and SP lengths, as obtained from Swiss-Prot.

Length (Amino Acids)	Signal Peptide	Protein
Mean ± StDev	23.5 ± 6.2	422.5 ± 454.1
Minimum	10	21
Median	22	323
Maximum	69	13076

I: Signal peptides generated from randomized protein sequences

The following sequences were drawn randomly from *Bacillus*, bacterial, and a uniform amino acid distribution, respectively to serve as inputs to the trained Transformer model.

- *Bacillus* DTTRFTAIEIFTTHWSGMVLFMSIIIVVFFSIGDSGVGDANIF YHQARLFKADKIANVQTLKGVYSNPLDYYFNHDLPRSGRLNLGNSH HIQLNKLQILF
- Bacterial SEVHPENTYQEGLPGIKALRIQFMTATIVHEGSTIDEVVRK KALTQTTCIPYLIAASIIVSSGEDMHIIVIKHMAQQVLPKLISCAMEHF VGPSHAEASE
- Uniform DIVPSLYPPAKSDERKVELTSKQSELNSENLKATQQILKDG YATPGRVRKIDAETKKQLNLIKHTYRTVRKQEDKGVYQASNPRNFK FIVPQLVQYKGRN

Using the mixed input beam search for these three inputs, as described in the main text, generated the following three sequences:

- MKLLVLLILILIPVVAWA
- MNWLVIMFAIGLACSTSLA
- MEHTLRILVCSLLFTVAIWG

II: Comparing various generation approaches

In this section, we compare four methods of generating SP sequences for their efficacy: (1) a profile Hidden Markov Model (pHMM), (2) a heuristic-based generation approach, (3) a variational autoencoder (VAE) [35] trained on just SP sequences, and (4) the Transformer-based approach. In the first round of comparison, we query the SignalP 5.0 server with these SP sequences prepended to protein sequences to obtain SignalP's probability of functioning.

1. pHMM: We first generated an alignment with Clustal Omega [42] of SPs from our training. We used 4000 randomly selected SPs, as Clustal Omega is limited to 4000 sequences. From this point on, we use default settings in S. Eddy's HMMER package [34]. From the Clustal Omega alignment, we then use HMMER's *hmmbuild* to build a profile HMM. From this profile HMM, we use HMMER's *hmmemit* to generate protein samples. As HMMs typically follow the *first* order Markov property, the HMMs do not have a concept of length, and can generate longer sequences. We generate 1024 constructs by emitting 1024 SP sequences from *hmmemi*t, each of which is prepended to one of the 41 enzymes tested.

2. Heuristics: We follow the following heuristics from a recent review by Owji, Hajar, *et al* [25]. The N-region (1-6 residues) contains at least 1 positive residue (KR). The H-region (7-15 residues) contains hydrophobic residues (AILMFWYV). The C Region (3-7 residues) contains uncharged residues (STNQAVILMFYW) followed by an AXA motif, where X is not in C/P/A. 768 SPs were generated by heuristics and randomly matched to one of the 41 input proteins.

3. VAE: We train a VAE [35] with fully connected layers and ReLU activation on SP sequences. The sizes of the hidden layers are 400 and 20 each for the mean and variance of the latent encoding. We then randomly sample 1024 SP sequences, each of which is prepended to one of the 41 enzymes tested.

4. Transformer: As described in the main text.

We then query these constructs with SignalP v5.0 to obtain the probability of containing a Signal Peptide in gram-positive hosts [43]. The results are shown visually in the figure below, and summarized in its accompanying table.

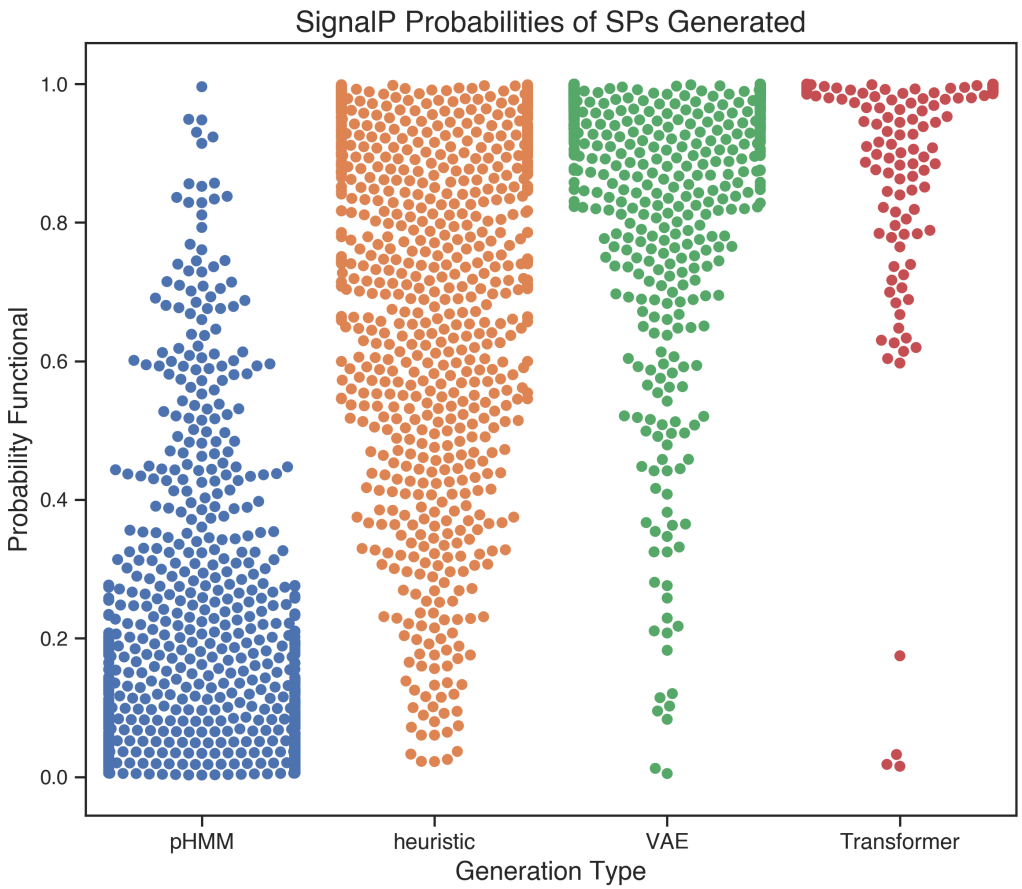


Figure 4.5: Probability of generated sequences as predicted by SignalP 5.0 The sequences are generated by either a pHMM, heuristic-based approach, VAE, or Transformer model. Results are summarized in the table below.

Table 4.6: Distribution of SignalP Probabilities for sequences generated by a profile Hidden Markov Model, heuristics, a variational autoencoder, and a Transformer model.

Method	Mean SignalP Probability
pHMM	15.6% ± 19.2%
heuristic	70.8% ± 25.3%
VAE	**92.4% ± 15.2%**
Transformer	90.4% ± 17.1%

The VAE and Transformer significantly outperform the other two methods (p-value < 10^{-20}), but are not statistically different from each other (p-value = 0.251). As both are predicted to function with high probability by SignalP, we turn to other methods of analysis below.

First, we compare the position-normalized log-likelihood on an identical, withheld validation set of 5707 SP sequences for the VAE and Transformer, which are comparable between the two methods, although the VAE is more accurate by this metric.

Table 4.7: Comparing normalized log likelihood on withheld validation set between a trained VAE and Transformer.

Method	Log likelihood on Validation Set
VAE	**-2.86**
Transformer	-3.01

However, in closer examination of the sequences generated by the VAE and Transformer, an issue of repeated residues begins to emerge:

Table 4.8: Comparing sample sequences generated by a VAE versus Transformer

Sample VAE Sequences	Sample Transformer Sequences
MRKRLLALALAALSLLLLLSFGVKALAGSGA	MKFTQAVLSLLGSAATALA
MRLLLLLLLVLLLAAPAPPGLS	MKLKKLGVILAICLGISSTFA
MKLLLLLVTLLTSVLALQA	MRVLSATAFLALLAFGLSGATA
MRLLLLALLAAAAVALASA	MLFKSVLLALASAGVAVNA
MASSSSSLFVVLAVLLLLLLLTLSSA	MNISIFVGKLALAALGSALVA

Namely, the hydrophobic region appears to be particularly saturated in repeating residues for VAE sequences. For a more comprehensive comparison, we determined the average length of substrings with maximum R repeating residues (e.g.: an example of $R = 1$ is LLLLLLL, $R = 2$ is LLLILIL, $R = 3$ is LALLAI).

Table 4.9: Comparing longest repeating substrings of R unique characters between a trained VAE and Transformer.

	Average longest substring from VAE Seqs	Average longest substring from Transformer Seqs
$R = 1$	3.9 ± 1.9	2.1 ± 0.6
$R = 2$	7.2 ± 3.0	4.0 ± 1.0
$R = 3$	10.7 ± 3.5	6.3 ± 1.5

As biological sequences rarely contain such long repeats, we elect to proceed with the Transformer generated sequences in experimental validation.

III: Functionality classification from experimental validation

SP-enzyme constructs were generated stochastically and tested in biological replicates. Thus, some random SP-enzyme constructs contain the same SP and enzyme but are not part of the same set of biological replicates.

An SP-enzyme construct is classified as functional if any set of biological replicates satisfies two conditions (p-value and effect size):

1. The p-value for the set of replicates compared to the negative controls (constructs without either an SP or enzyme) was less than 0.05 in a two-sided t-test with unequal variance for two independent samples of scores. Biological replicates are grown from the same colony, after which they are processed separately. The null hypothesis is that these two independent samples have identical expected values. The scipy.stats implementation (ttest_ind) was used. (p-value)

2. The difference between the mean of the replicates and the mean of the incorrect constructs is greater than 2 times the difference between the mean of the incorrect constructs and the highest measured activity of the incorrect constructs. (effect size)

The second condition is incorporated to gain some concept of effect size: the activity of the SP-enzyme construct must be high, not just statistically significant.

In the following plots, constructs classified as functional are to the left of the "incorrect constructs", which are shown in black. For functional constructs, some sets of replicates (picked from the same colony) would have been classified as nonfunctional. These are shown in blue. The functional sets of replicates are shown in green. Finally, constructs classified as nonfunctional are shown in grey. Summary statistics and notes highlighting important points are also provided.

All generated Signal Peptides can be found in Supplemental Section VII.

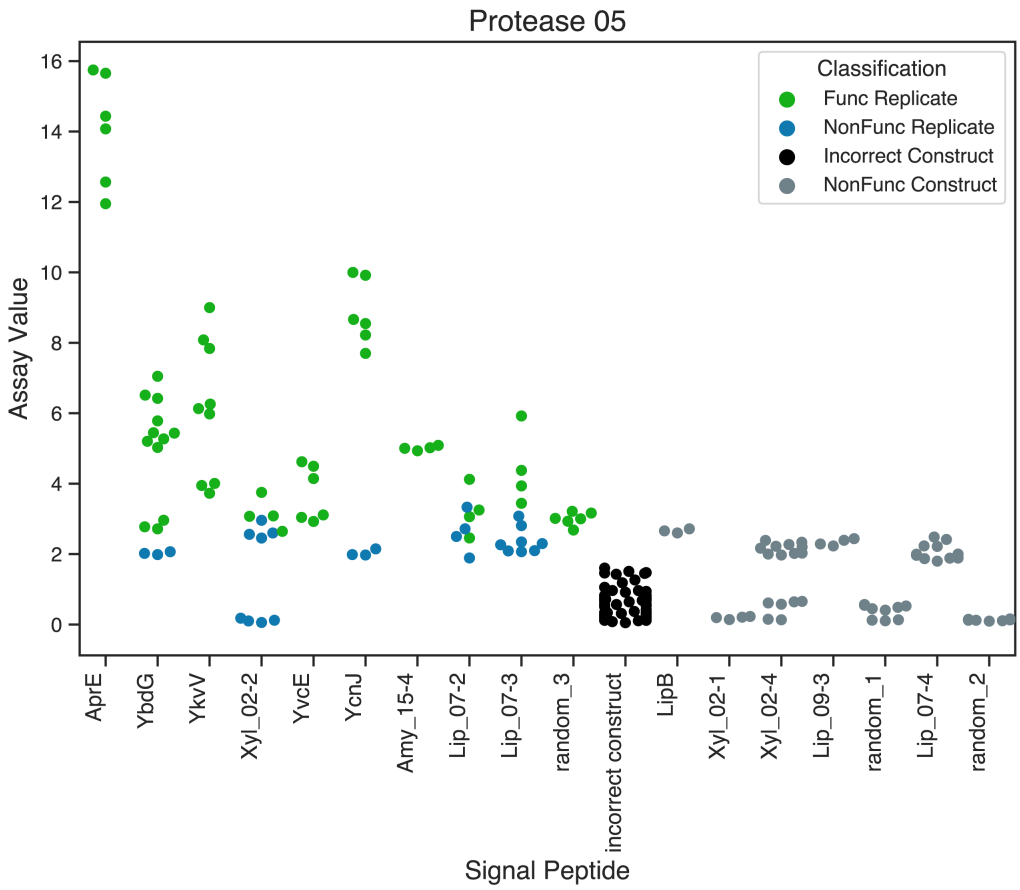


Figure 4.6: Experimental validation of Protease 5.

Construct Functionality Classification Summary

0.33 : 1 out of 3 constructs with SPs generated for random inputs functional*

0.83 : 5 out of 6 constructs with natural SPs functional

0.50 : 4 out of 8 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

*The SP sequence for the control generated for a randomized input that was classified as functional is **MEHTLRILVCSLLFTVAIWG**.

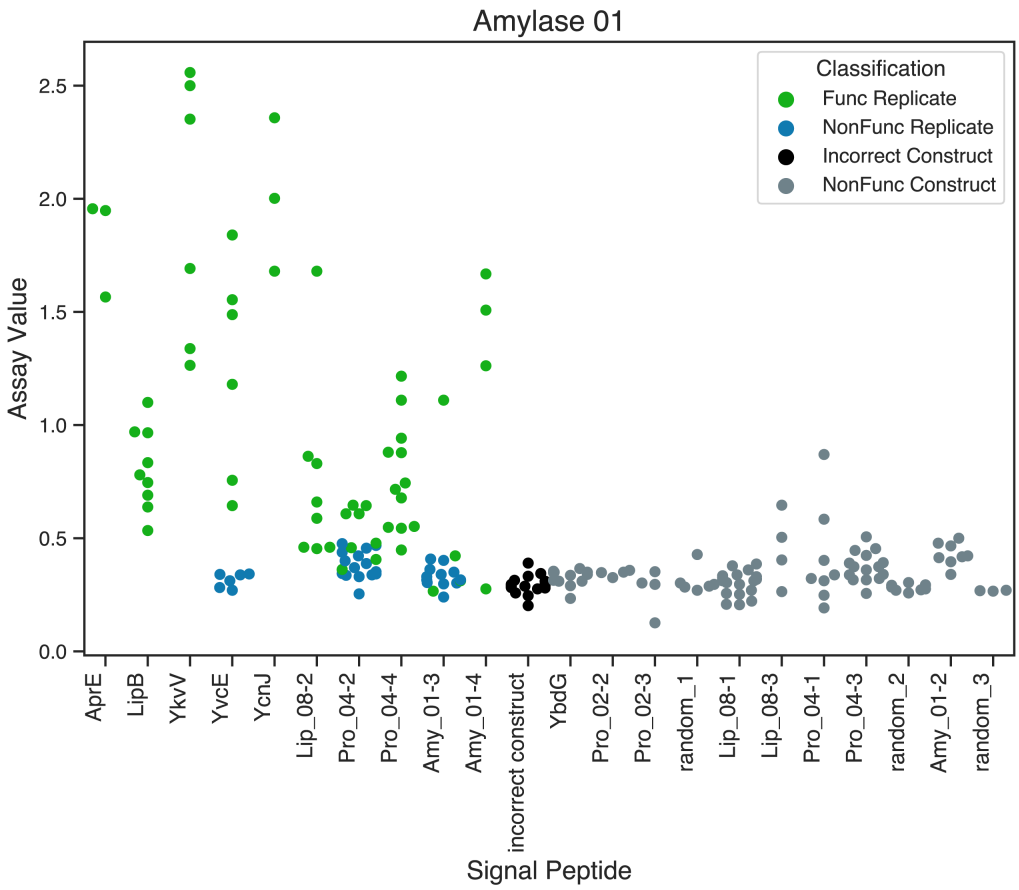


Figure 4.7: Experimental validation of Amylase 1.

Construct Functionality Classification Summary

0.00 : 0 out of 3 constructs with SPs generated for random inputs functional

0.83 : 5 out of 6 constructs with natural SPs functional

0.42 : 5 out of 12 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

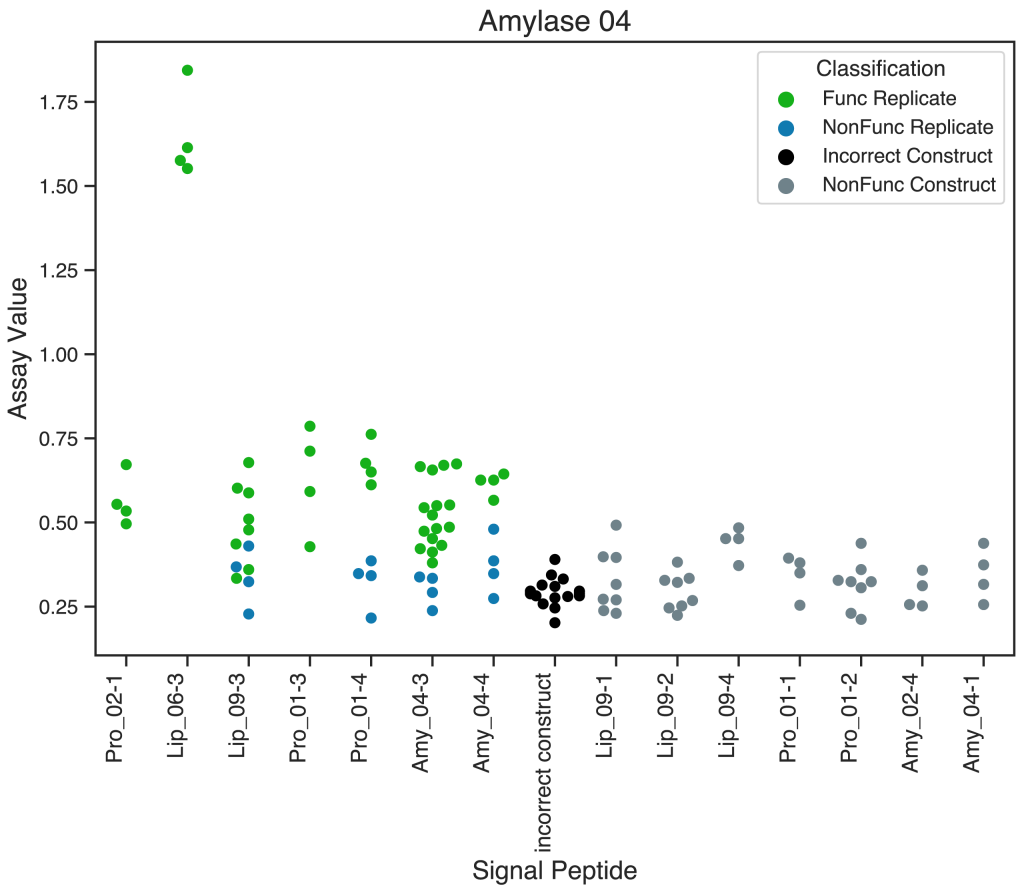


Figure 4.8: Experimental validation of Amylase 4.

Construct Functionality Classification Summary

No constructs with SPs generated for random inputs tested.

No constructs with natural SPs tested.

0.50 : 7 out of 14 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

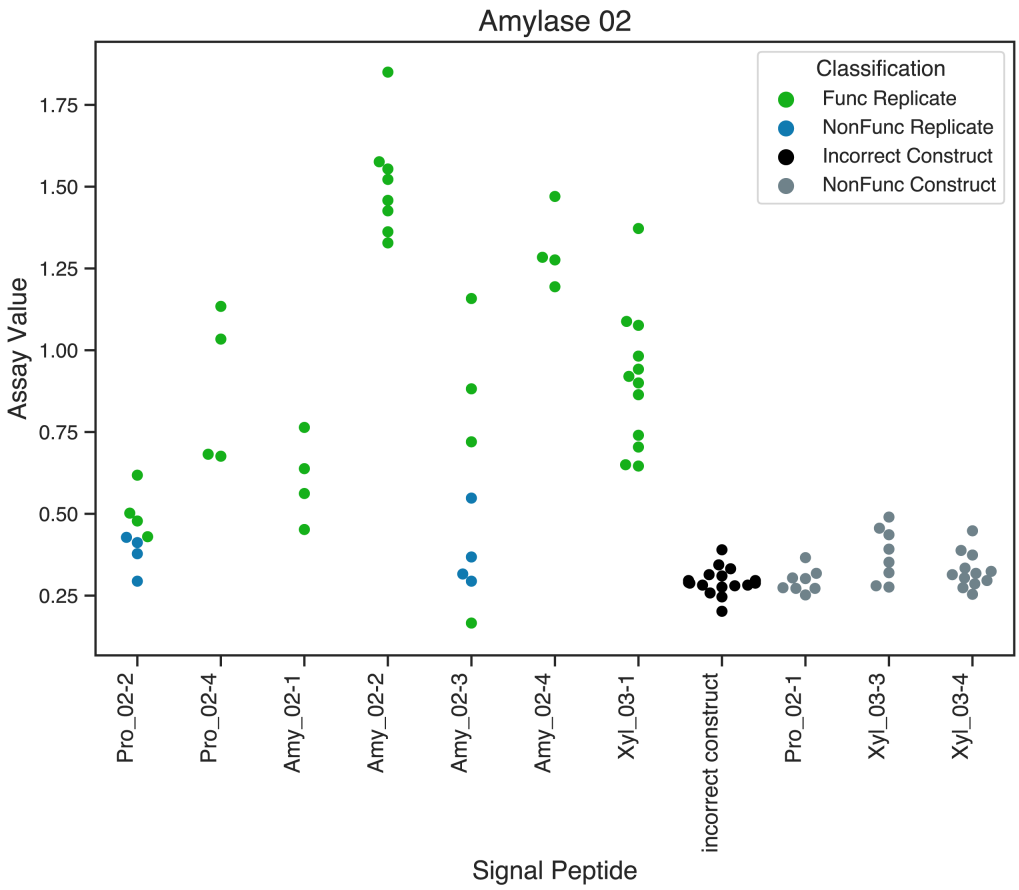


Figure 4.9: Experimental validation of Amylase 2.

Construct Functionality Classification Summary

No constructs with SPs generated for random inputs tested.

No constructs with natural SPs tested.

0.70 : 7 out of 10 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

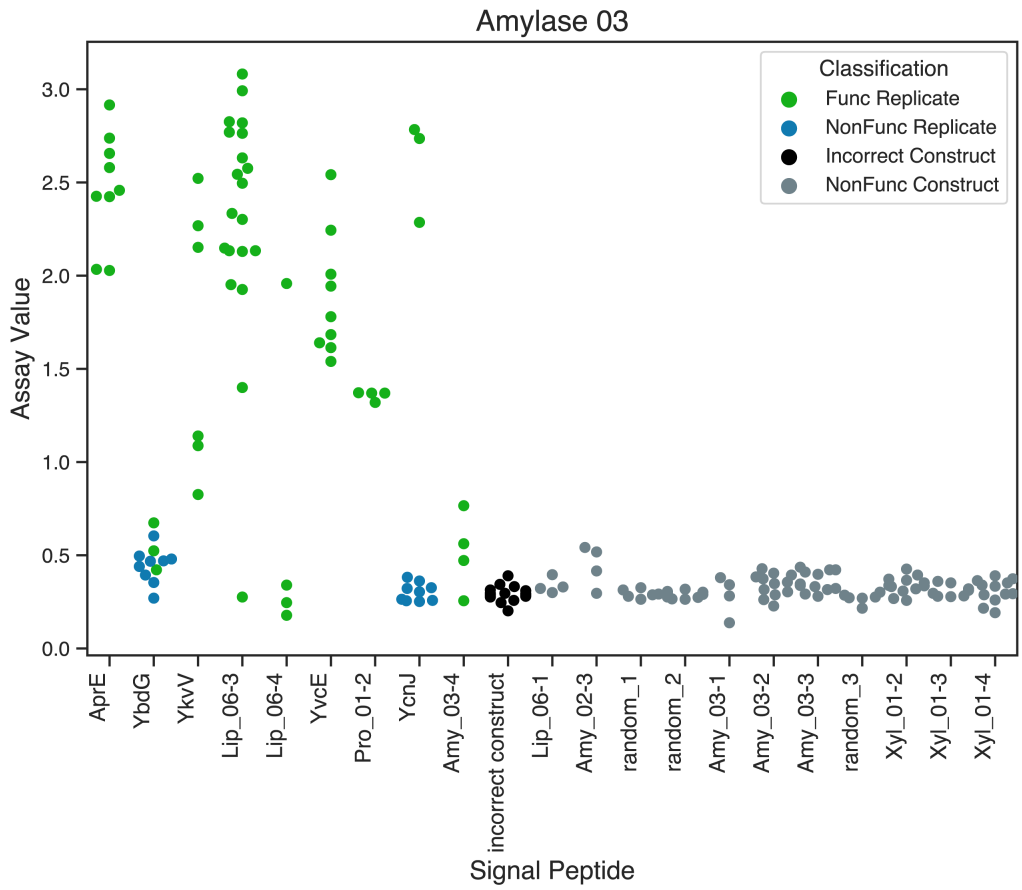


Figure 4.10: Experimental validation of Amylase 3.

Construct Functionality Classification Summary

0.00 : 0 out of 3 constructs with SPs generated for random inputs functional

1.00 : 5 out of 5 constructs with natural SPs functional

0.33 : 4 out of 12 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

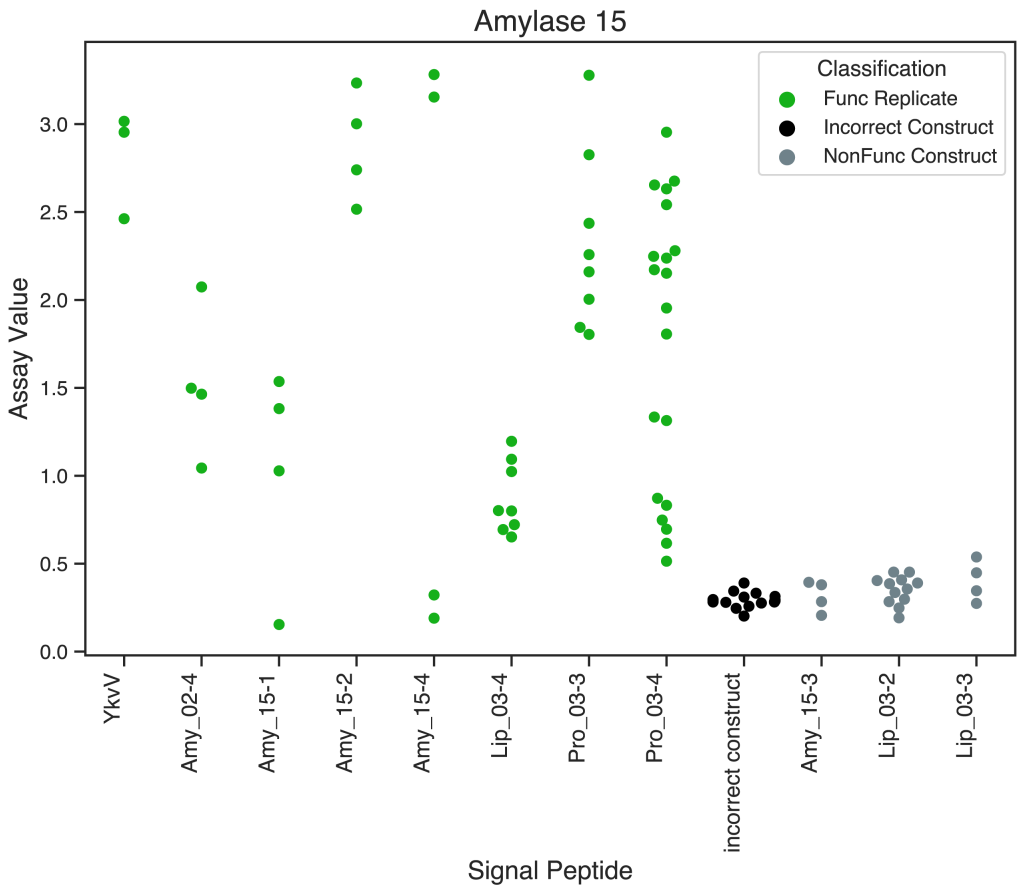


Figure 4.11: Experimental validation of Amylase 15.

Construct Functionality Classification Summary No constructs with SPs generated for random inputs tested.

1.00 : 1 out of 1 constructs with natural SPs functional

0.70 : 7 out of 10 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

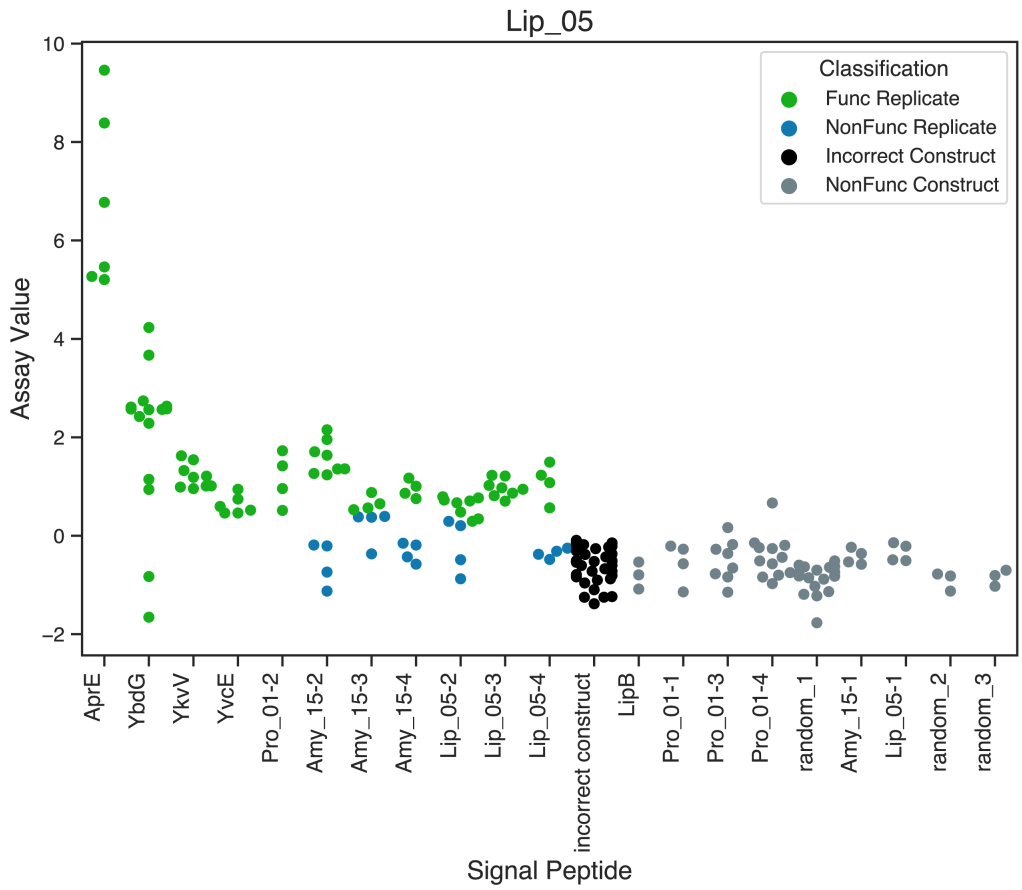


Figure 4.12: Experimental validation of Lipase 5.

Construct Functionality Classification Summary

0.00 : 0 out of 3 constructs with SPs generated for random inputs functional

0.80 : 4 out of 5 constructs with natural SPs functional

0.58 : 7 out of 12 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

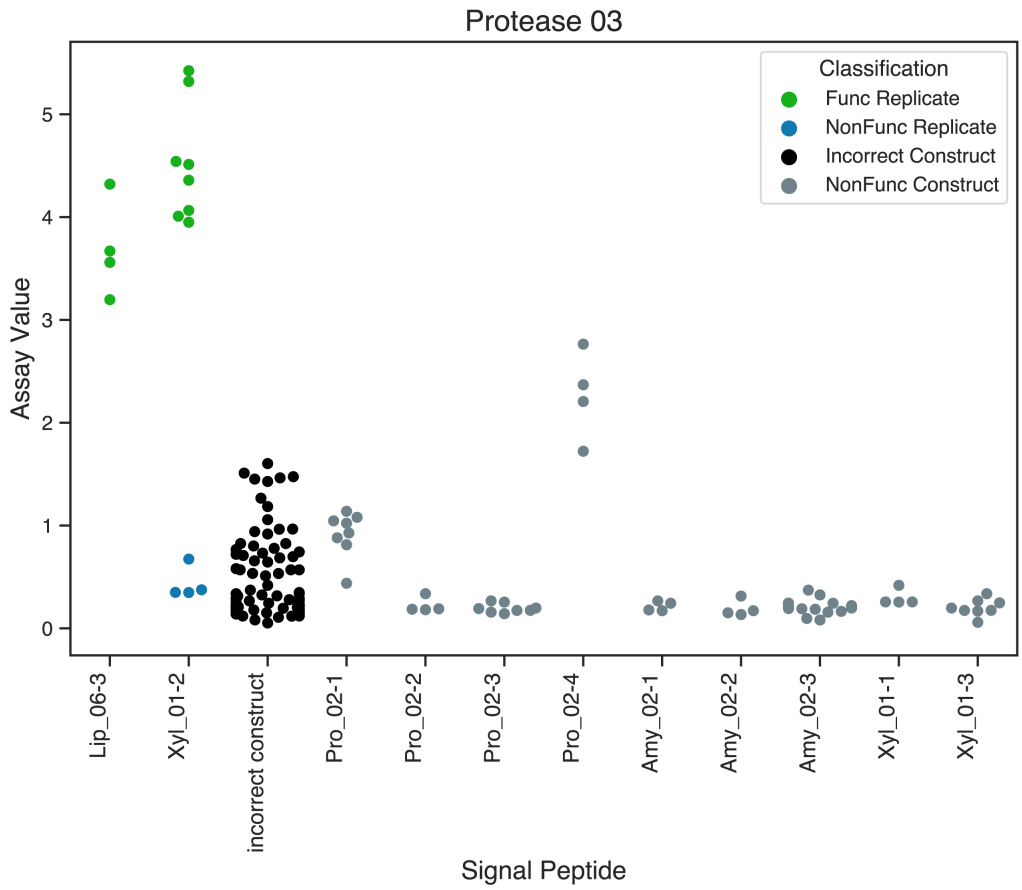


Figure 4.13: Experimental validation of Protease 3.

Construct Functionality Classification Summary

No constructs with SPs generated for random inputs tested.

No constructs with natural SPs tested.

0.18 : 2 out of 11 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

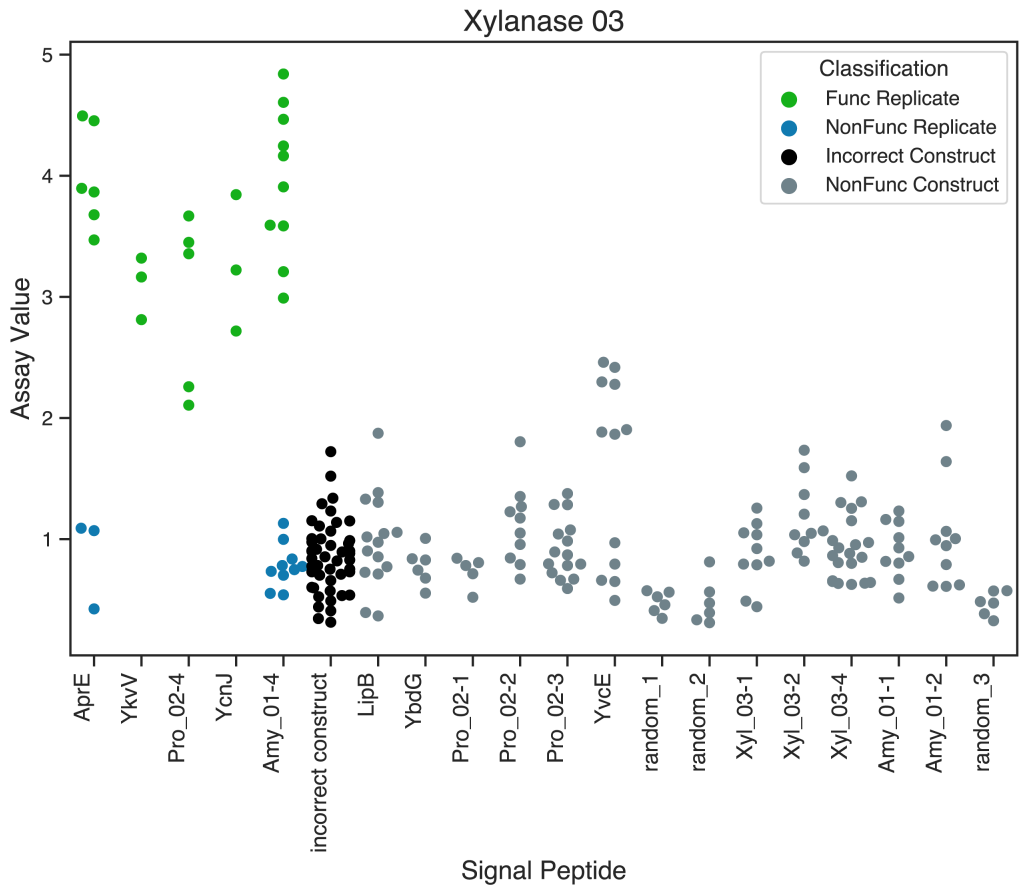


Figure 4.14: Experimental validation of Xylanase 3.

Construct Functionality Classification Summary

0.00 : 0 out of 3 constructs with SPs generated for random inputs functional

0.50 : 3 out of 6 constructs with natural SPs functional

0.20 : 2 out of 10 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

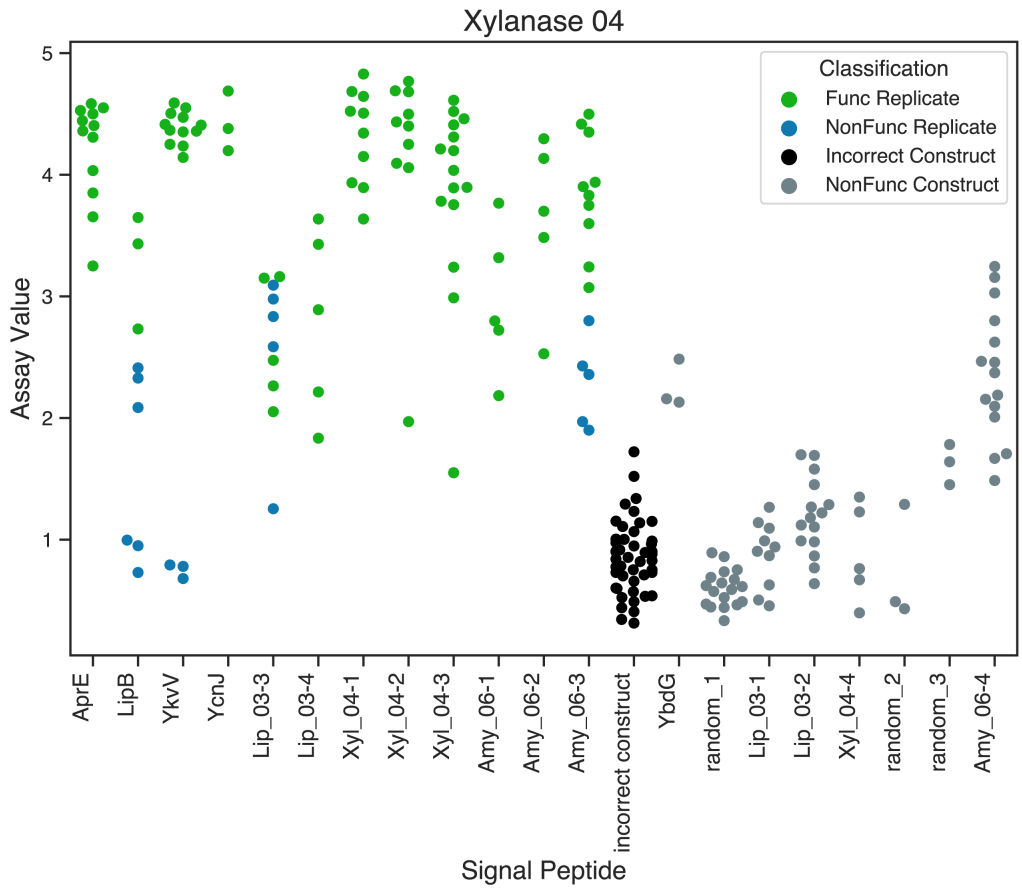


Figure 4.15: Experimental validation of Xylanase 4.

Construct Functionality Classification Summary

0.00 : 0 out of 3 constructs with SPs generated for random inputs functional

0.80 : 4 out of 5 constructs with natural SPs functional

0.67 : 8 out of 12 constructs with generated SPs functional

Assay information is provided in Supplemental Table 4.3.

IV: Activity assays at higher dilution

While the previous section shows activity data for identifying functional constructs, we found that some amylase and xylanase constructs were outside of the linear range (activity was too high) to make accurate comparisons between the high functioning constructs. To address this, we performed the assays at higher dilution (10x for amylase, 50x for xylanase) to obtain data in the linear range of the assay.

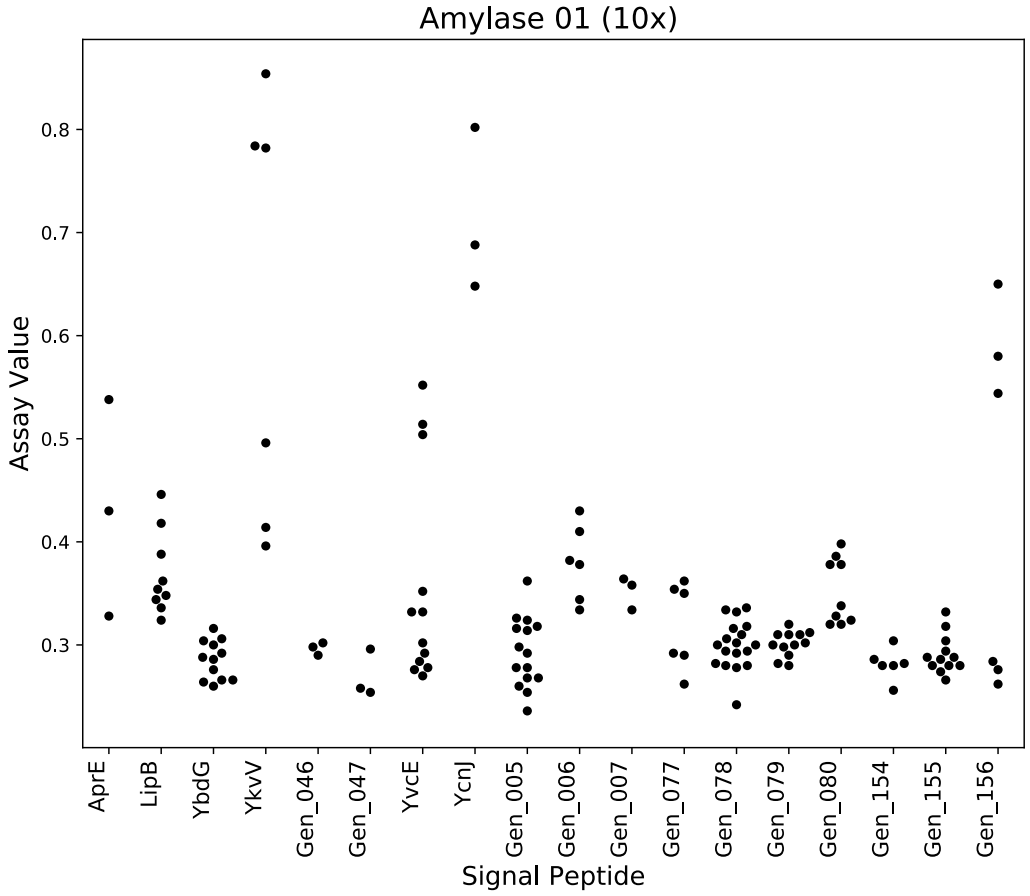


Figure 4.16: Experimental validation of Amylase 1 at higher dilution.

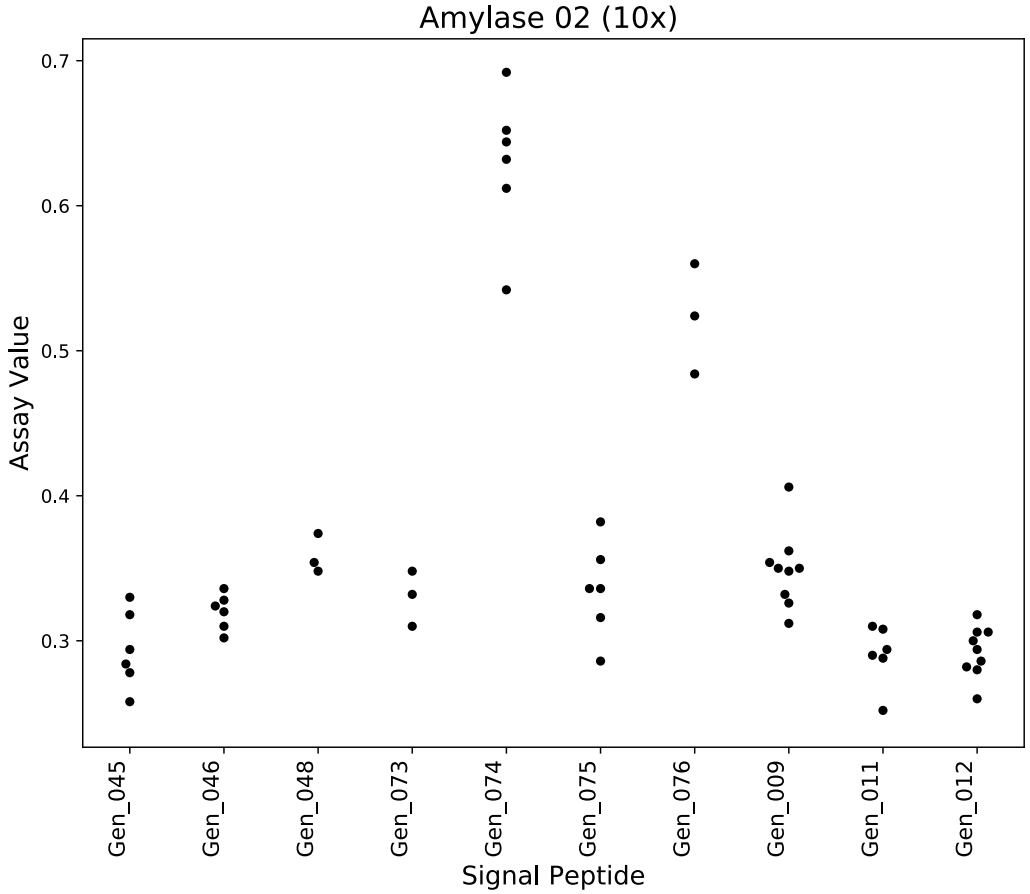


Figure 4.17: Experimental validation of Amylase 2 at higher dilution.

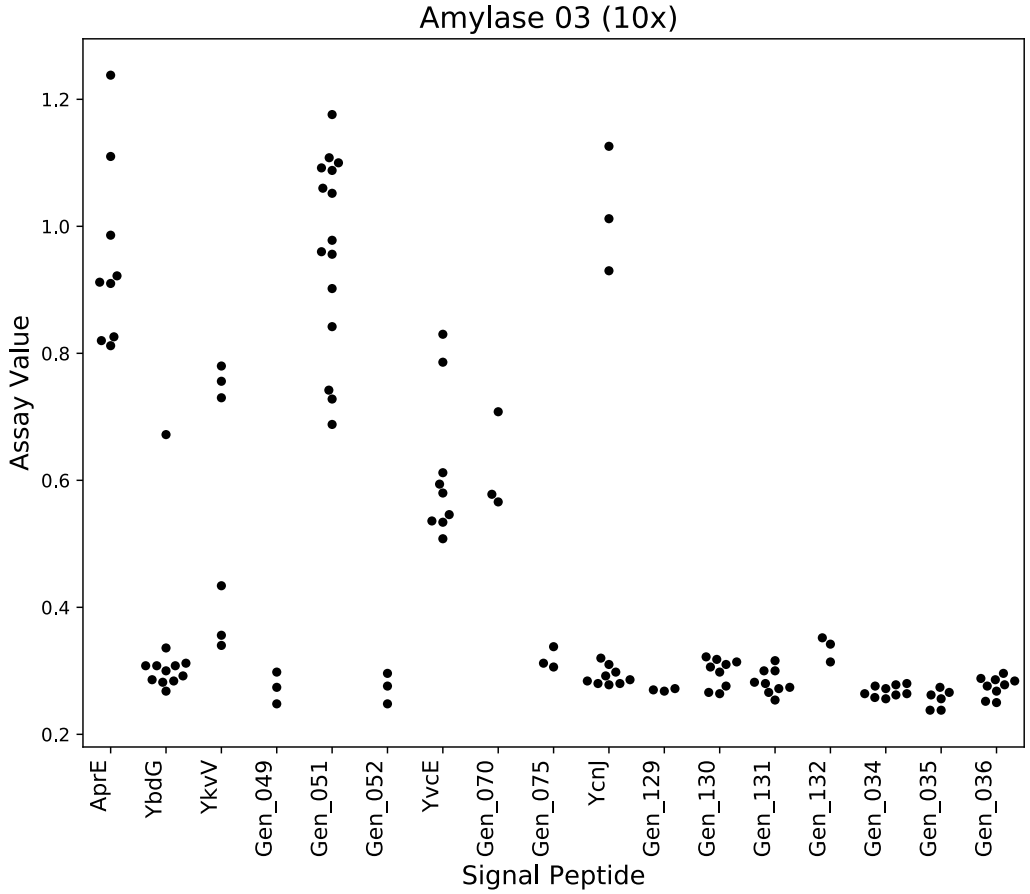


Figure 4.18: Experimental validation of Amylase 3 at higher dilution.

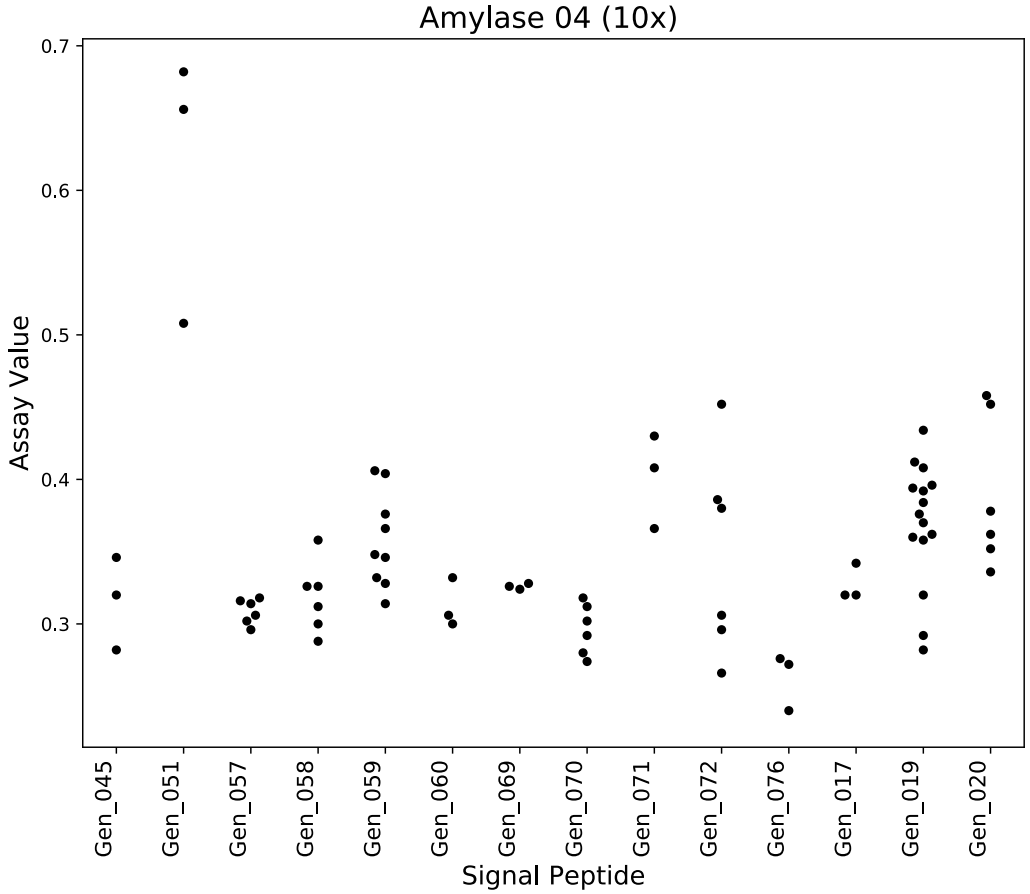


Figure 4.19: Experimental validation of Amylase 4 at higher dilution.

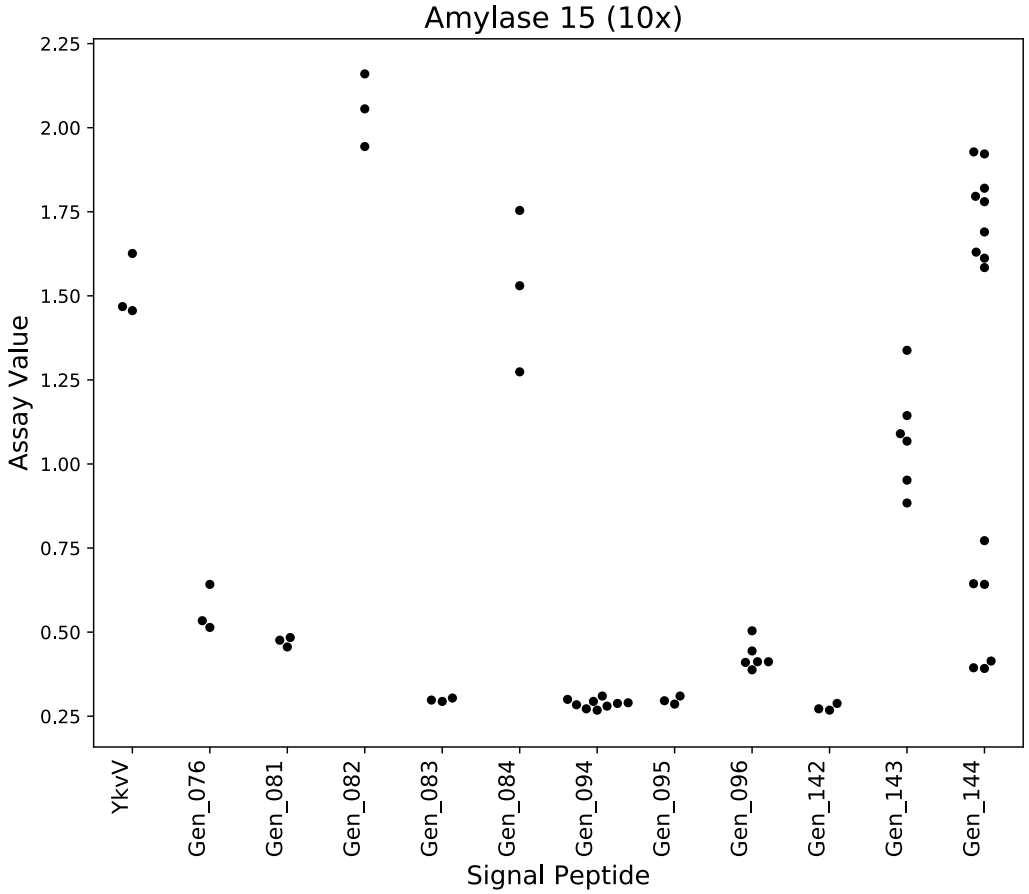


Figure 4.20: Experimental validation of Amylase 15 at higher dilution.

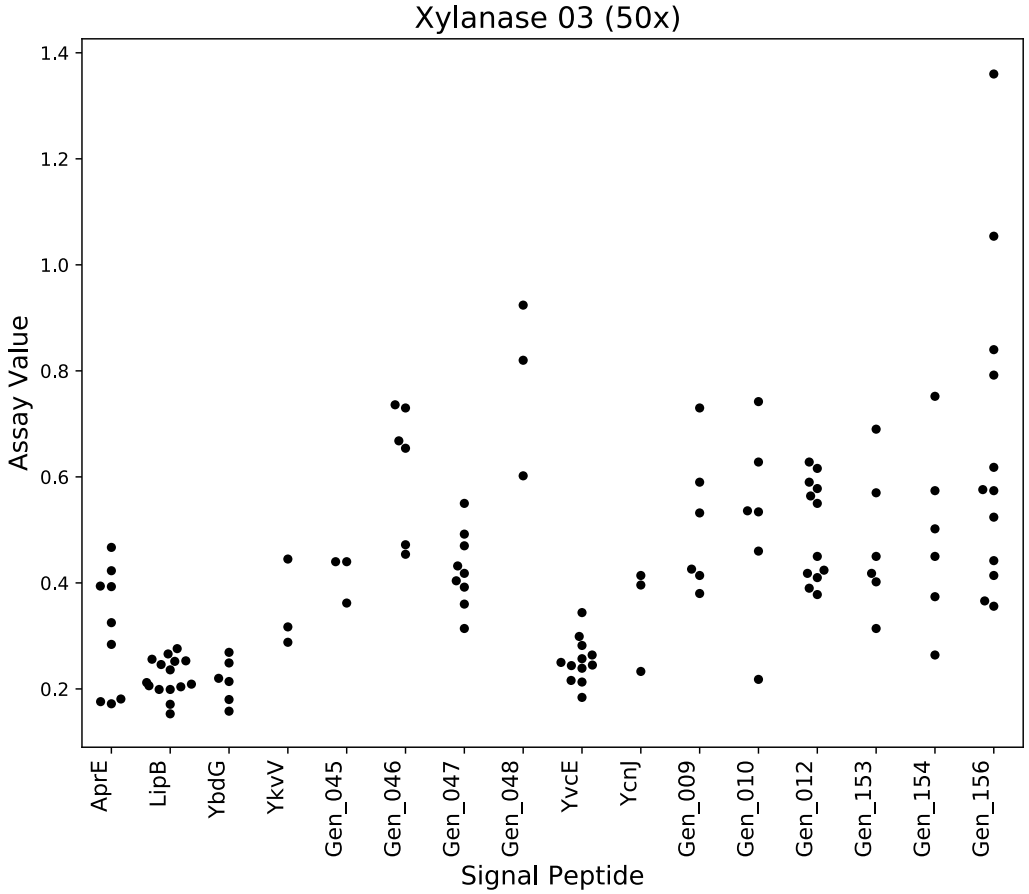


Figure 4.21: Experimental validation of Xylanase 3 at higher dilution.

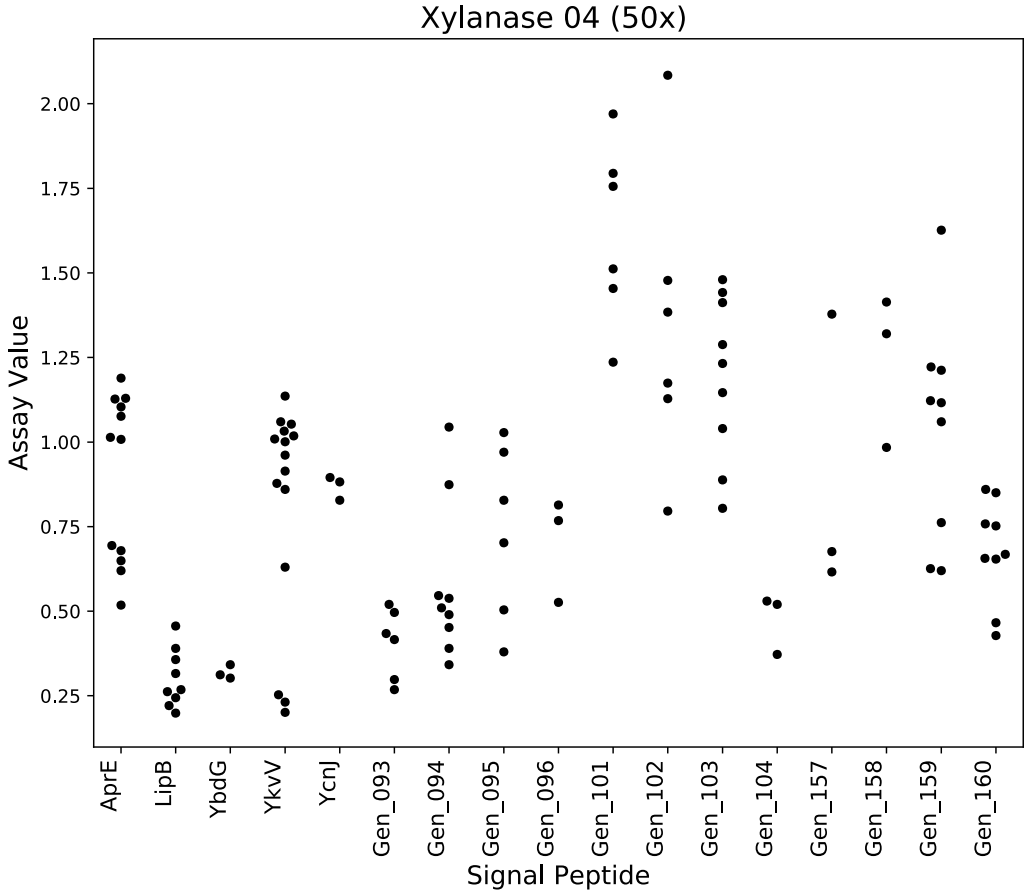


Figure 4.22: Experimental validation of Xylanase 4 at higher dilution.

V: Characteristics of functional vs nonfunctional generated SPs

We attempted to identify general characteristics of functional versus nonfunctional signal peptides that may help in discerning whether a signal peptide is functional or not. To do so, we separated the 27 signal peptides that are only present in functional constructs, and 30 signal peptides that are only present in nonfunctional constructs, and attempted to identify a distinguishing property. No property was identified for distinguishing between functional and nonfunctional constructs (p-value < 0.05) and SignalP 5.0 was also not able to discern between functional and nonfunctional generated SPs (ROC=0.59).

Table 4.10: Comparing physicochemical properties of functional vs nonfunctional generated SPs.

Protein Property	p-value
Length	0.9229
Hydrophobicity Index	0.387
Hydrophilicity Index	0.2126
Molecular Weight (MW)	0.7963
Average MW	0.117
Aromaticity	0.9935
Instability Index	0.9748
Isoelectric Point	0.3243

VI: All MSAs for functional SPs

MSAs for the top 10 closest matching natural Signal Peptides (by sequence identity) to each of the 43 generated SPs that were functional in any construct are provided. Alignments are sorted by closest sequence identity to the generated natural Signal Peptide (shown first).

All generated Signal Peptides can be found in Supplemental Section VII.

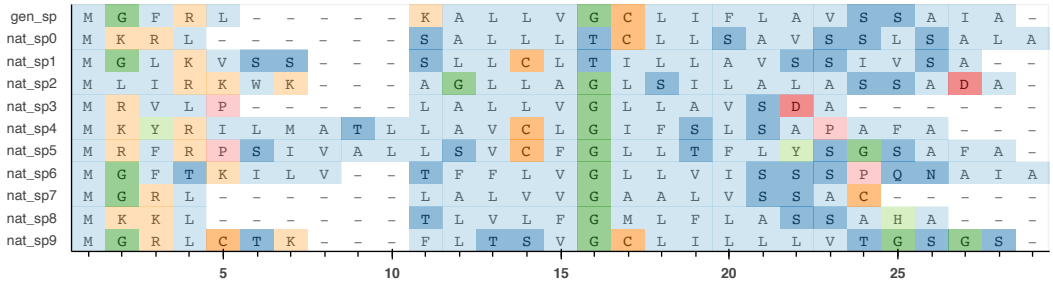


Figure 4.23: MSA for MGFRLKALLVGCLIFLAVSSAIA

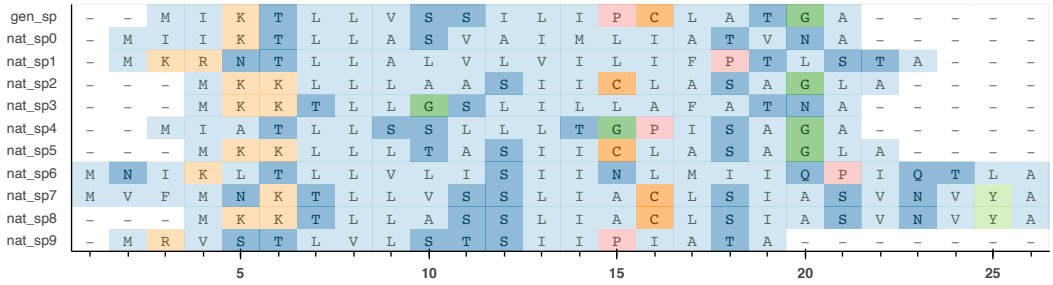


Figure 4.24: MSA for MIKTLLVSSILIPCLATGA

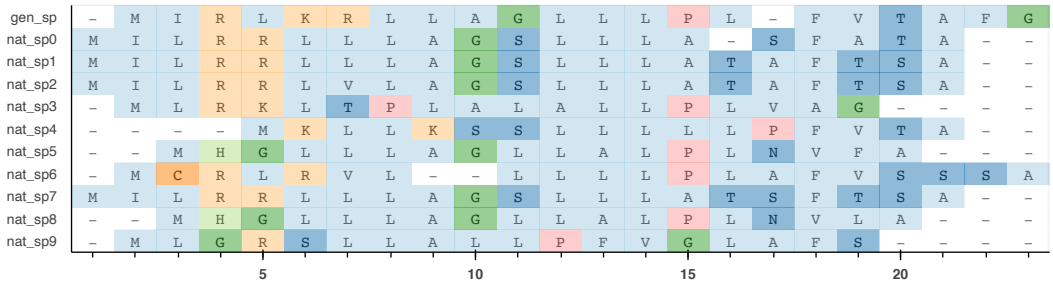


Figure 4.25: MSA for MIRLKRLLAGLLLPLFVTAFG

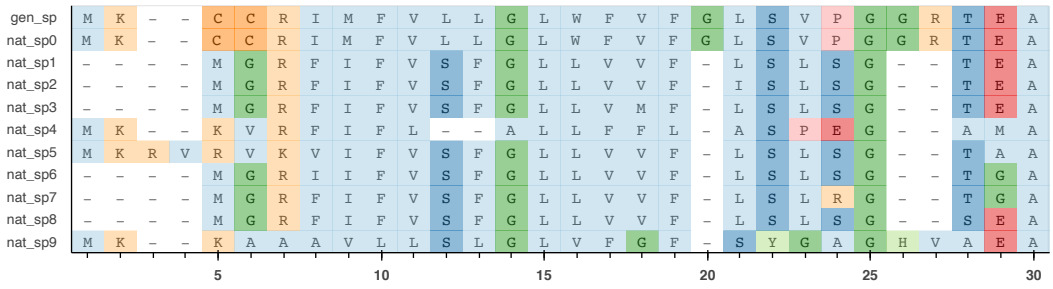


Figure 4.26: MSA for MKCCRIMFVLLGLWFVFGLSVPGGRTEA

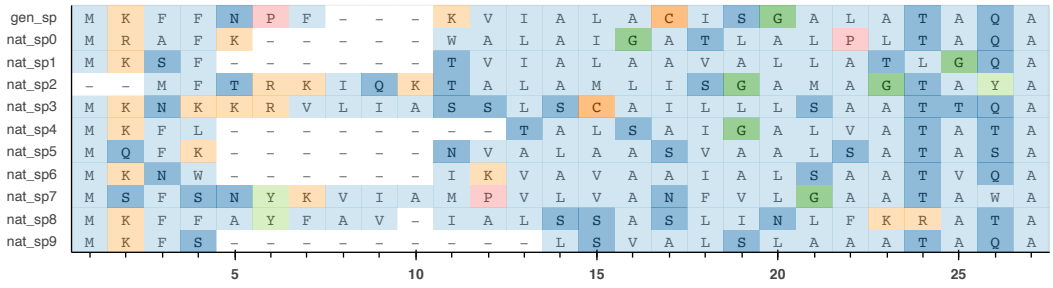


Figure 4.27: MSA for MKFFNPFKVIALACISGALATAQA

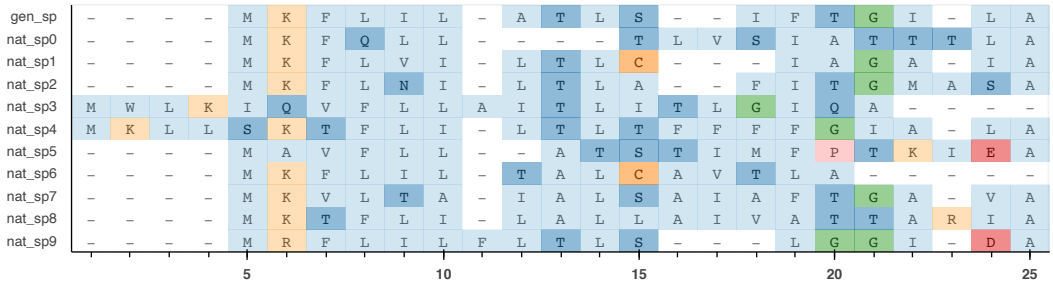


Figure 4.28: MSA for MKFLILATLSIFTGILA

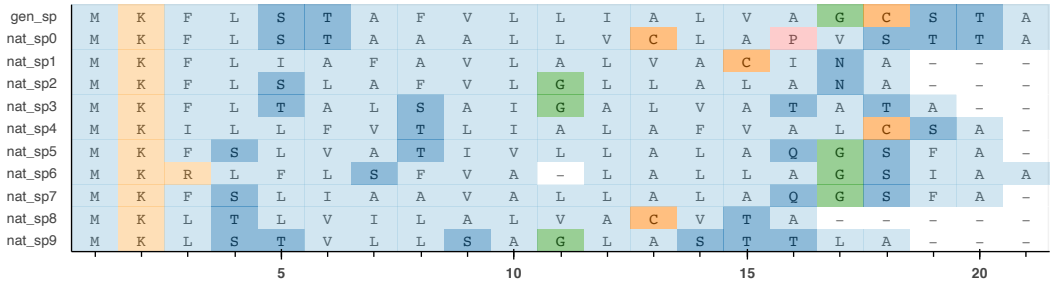


Figure 4.29: MSA for MKFLSTAFVLLIALVAGCSTA

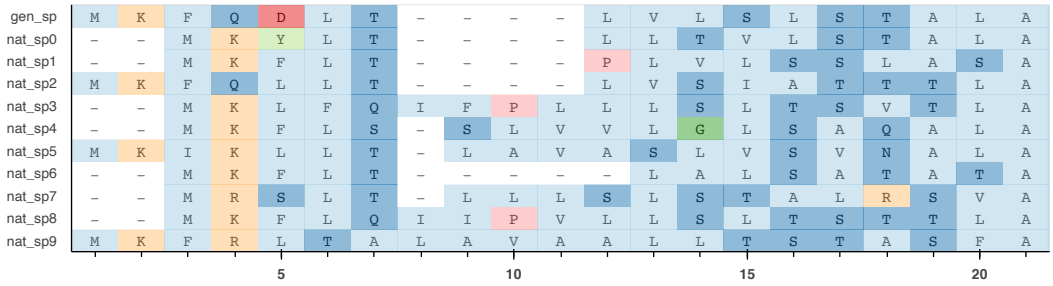


Figure 4.30: MSA for MKFQDLTLVLSLSTALA

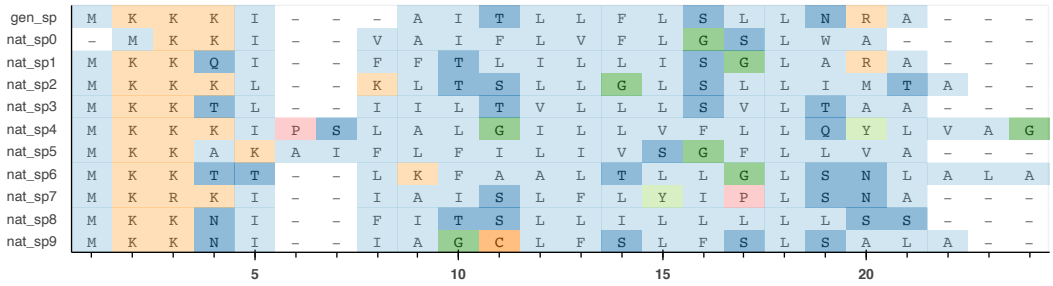


Figure 4.31: MSA for MKKKIAITLLFLSLLNRA

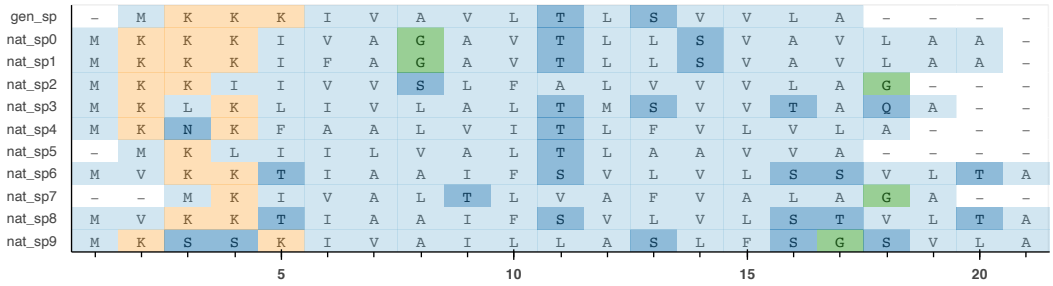


Figure 4.32: MSA for MKKKIVAVLTLSVVLA

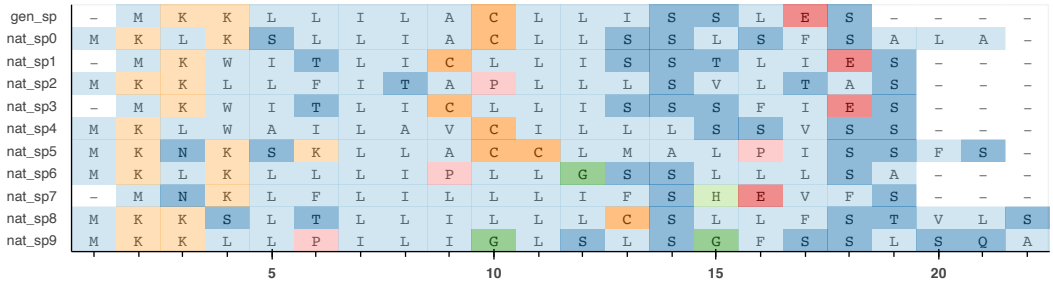


Figure 4.33: MSA for MKKLLILACLLISSLES

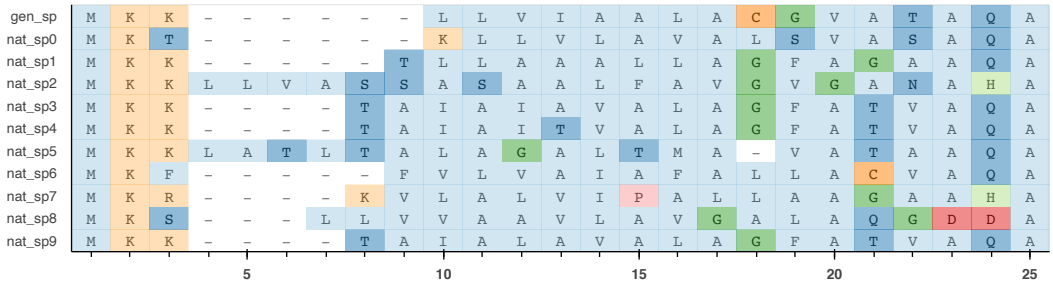


Figure 4.34: MSA for MKKLLVIAALACGVATAQA

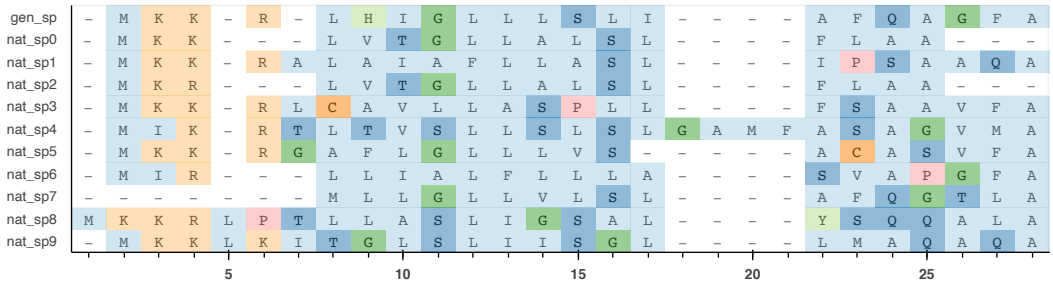


Figure 4.35: MSA for MKKRLHIGLLLSLIAFQAGFA

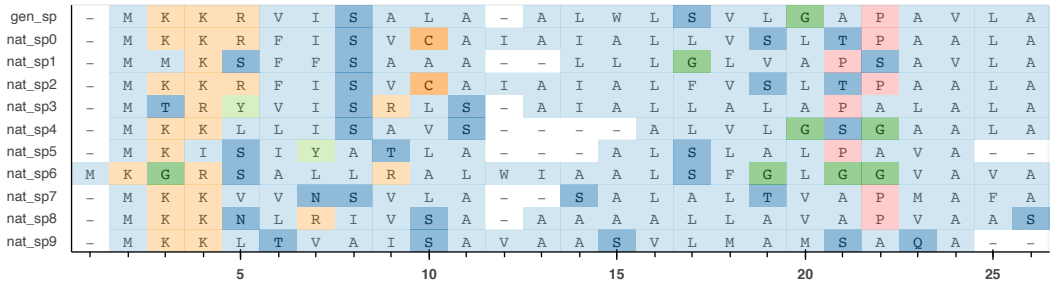


Figure 4.36: MSA for MKKRVISALAALWLSVLGAPAVLA

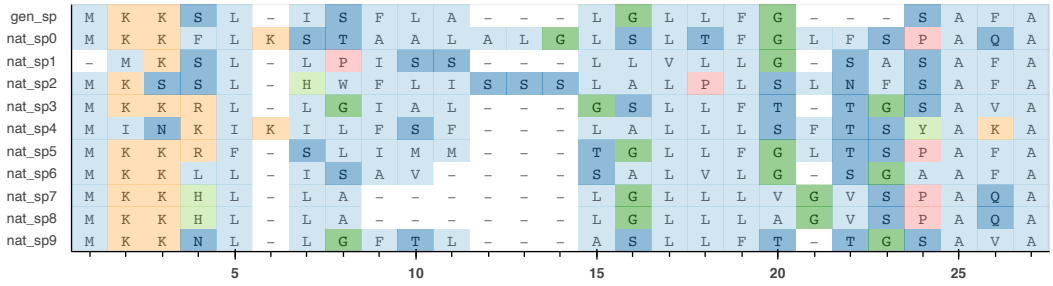


Figure 4.37: MSA for MKKSLISFLALGLLFGSAFA

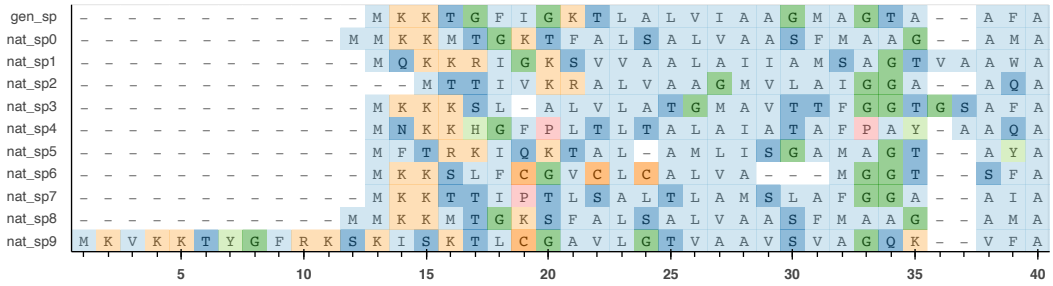


Figure 4.38: MSA for MKKTGFIGKTLALVIAAGMAGTAAFA

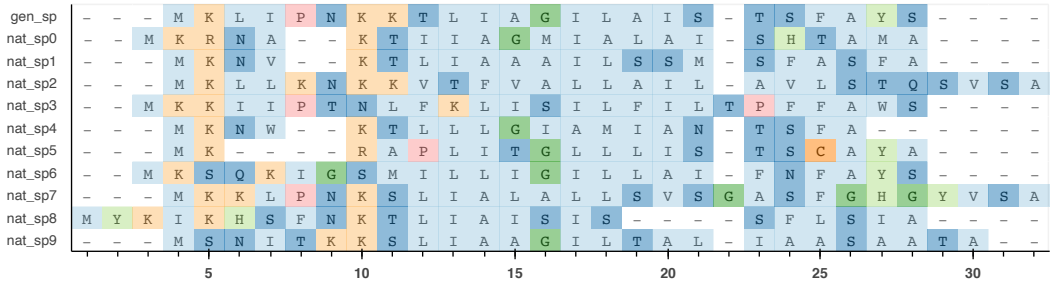


Figure 4.39: MSA for MKLIPNKKTLIAGILAISTSFAYS

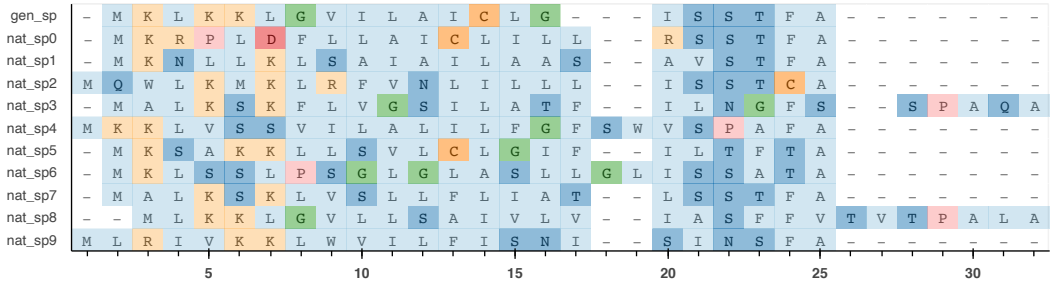


Figure 4.40: MSA for MKLKKLGVILAICLGISSTFA

Figure 4.41: MSA for MKLLTSFVLIGALAFA

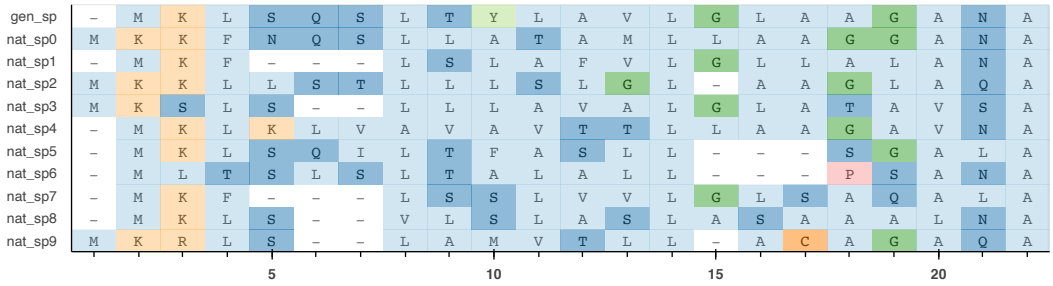


Figure 4.42: MSA for MKLSQSLTYLAVLGLAAGANA

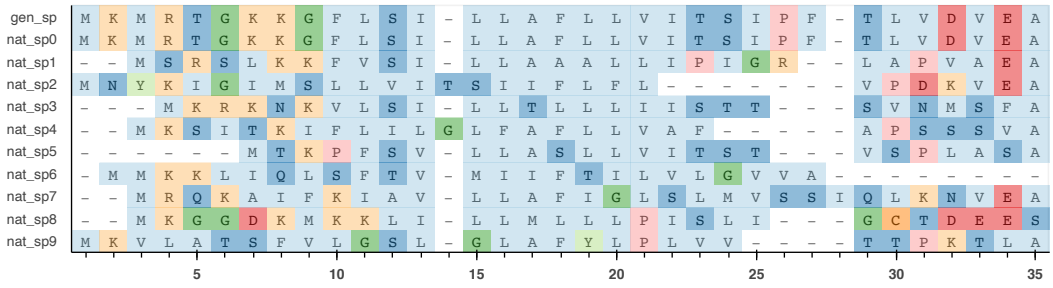


Figure 4.43: MSA for MKMRTGKKGFLSILLAFLLVITSIPFTLVDVEA

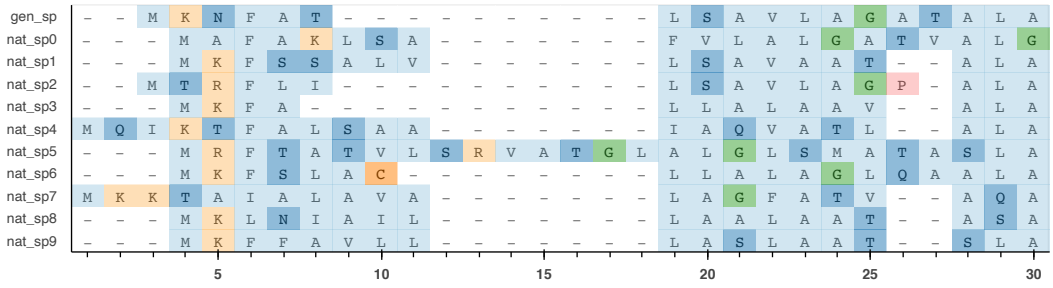


Figure 4.44: MSA for MKNFATLSAVLAGATALA

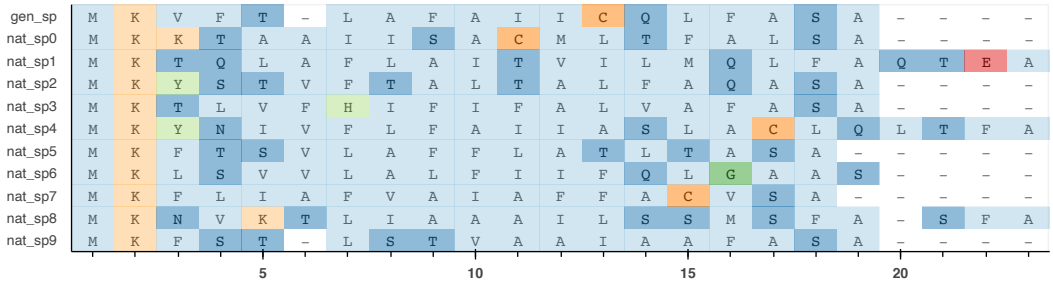


Figure 4.45: MSA for MKVFTLAFAIICQLFASA

Figure 4.46: MSA for MKVFTLAFFLAIIVSQA

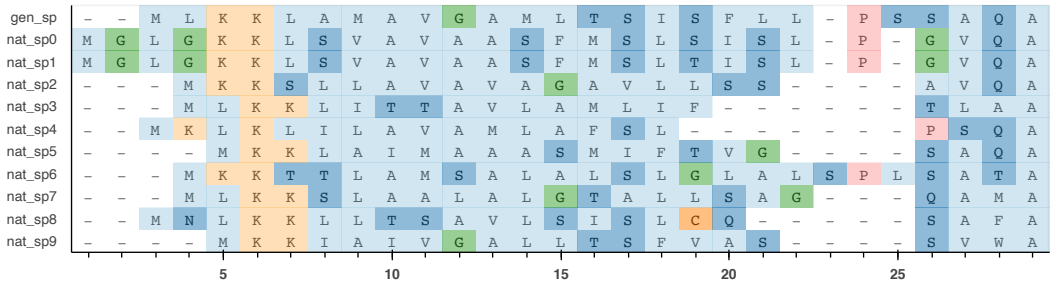


Figure 4.47: MSA for MLKKLAMAVGAMLTSISFLLPSSAQA

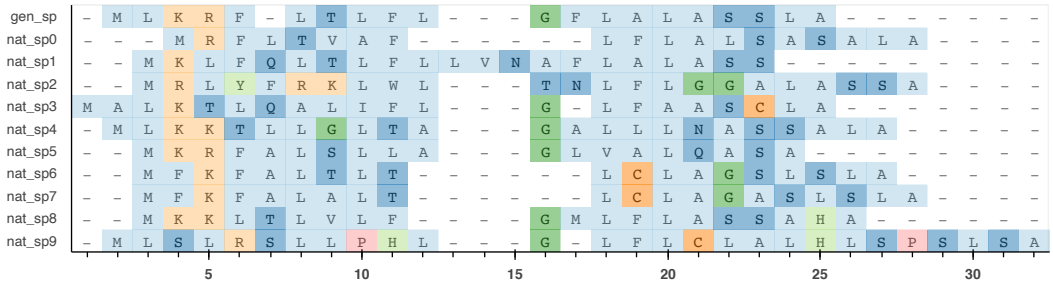


Figure 4.48: MSA for MLKRFLTLFLGFLALASSLA

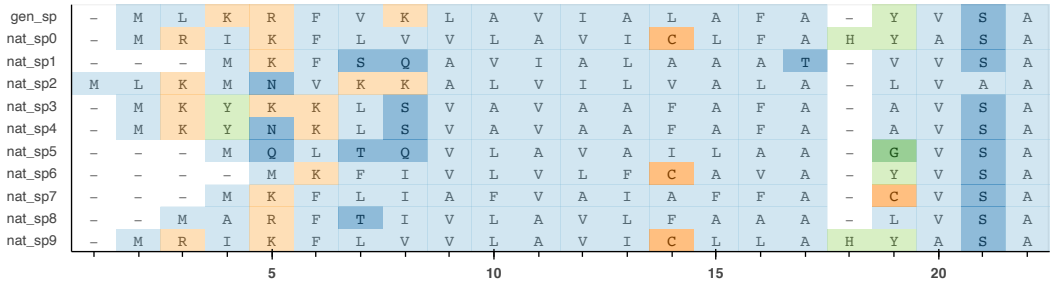


Figure 4.49: MSA for MLKRFVKLAVIALAFAYVSA

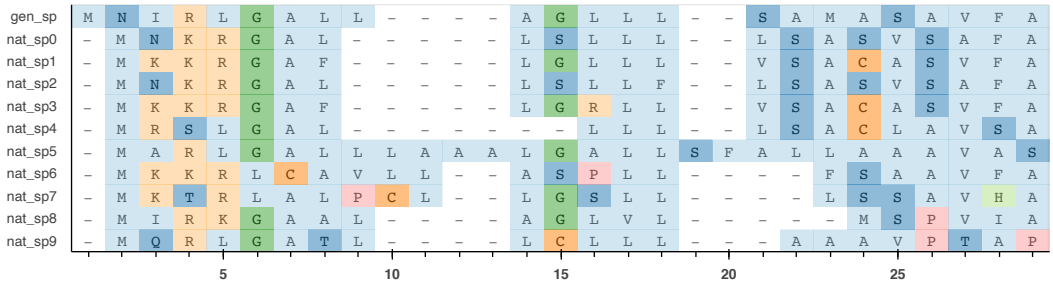


Figure 4.50: MSA for MNIRLGALLAGLLLSAMASAVFA

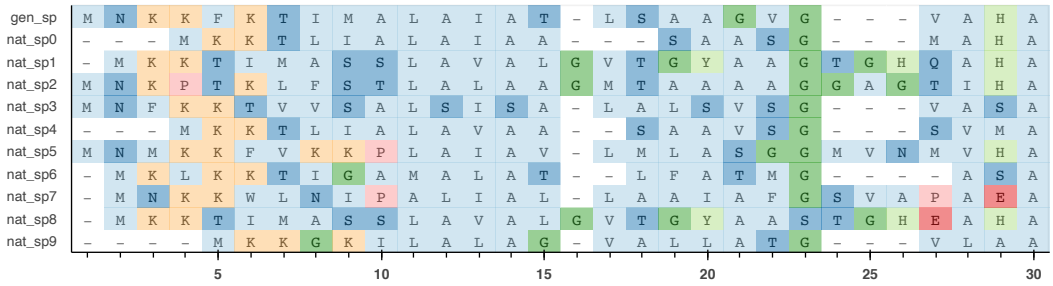


Figure 4.51: MSA for MNKKFKTIMALAIATLSAAGVGVAHA

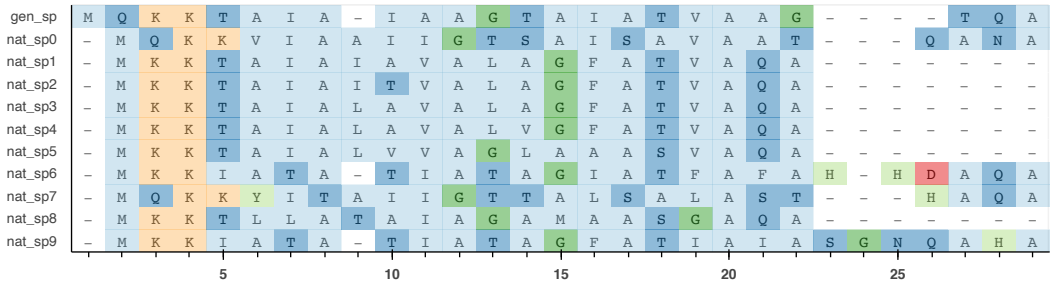


Figure 4.52: MSA for MQKKTAIAIAAGTAIATVAAGTQA

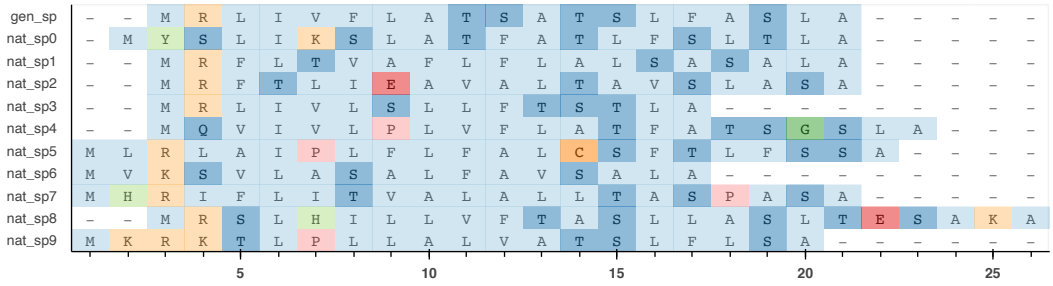


Figure 4.53: MSA for MRLIVFLATSATSLFASLA

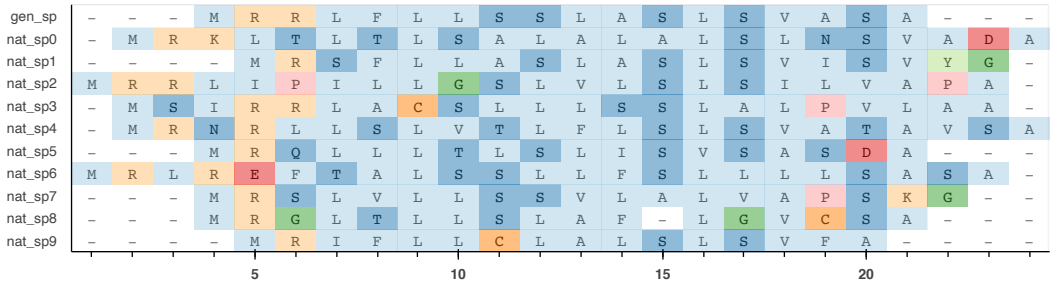


Figure 4.54: MSA for MRRLFLLSSLASLSVASA

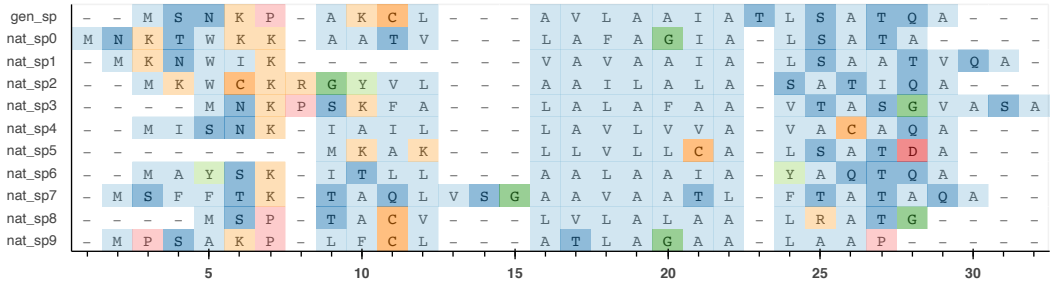


Figure 4.55: MSA for MSNKPAKCLAVLAAIATLSATQA

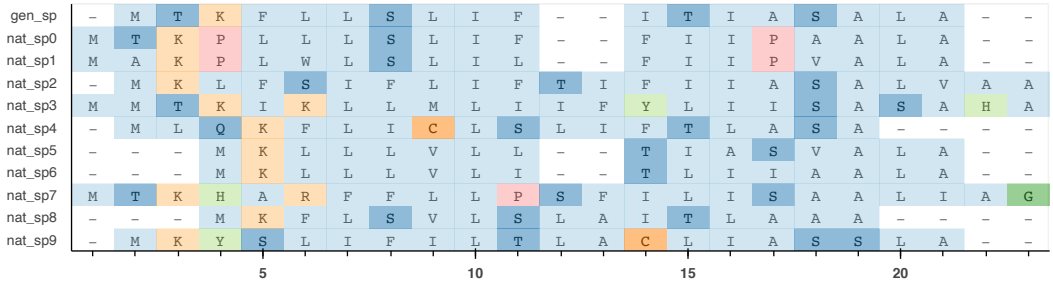


Figure 4.56: MSA for MTKFLLSLIFITIASALA

Figure 4.57: MSA for MTKLLAVIAASLMFAASTFA

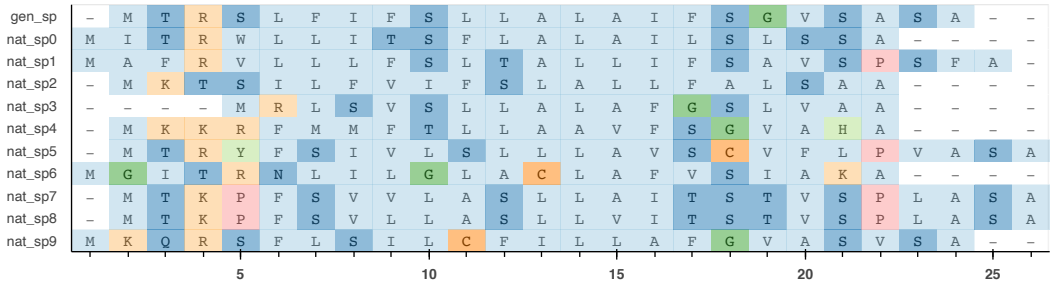


Figure 4.58: MSA for MTRSLFIFSLLALAIFSGVSASA

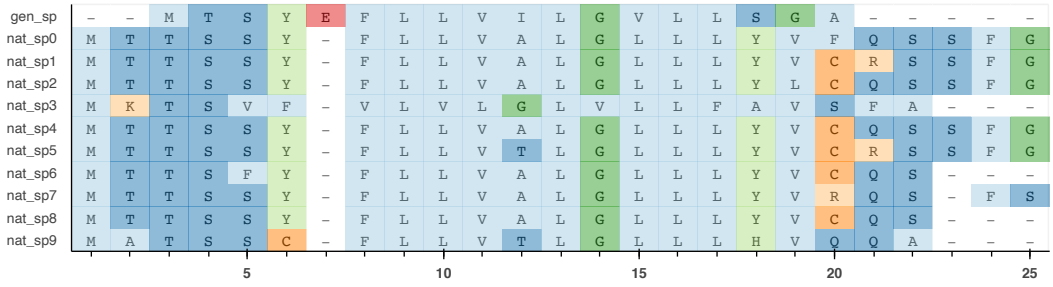


Figure 4.59: MSA for MTSYEFLLVILGVLLSGA

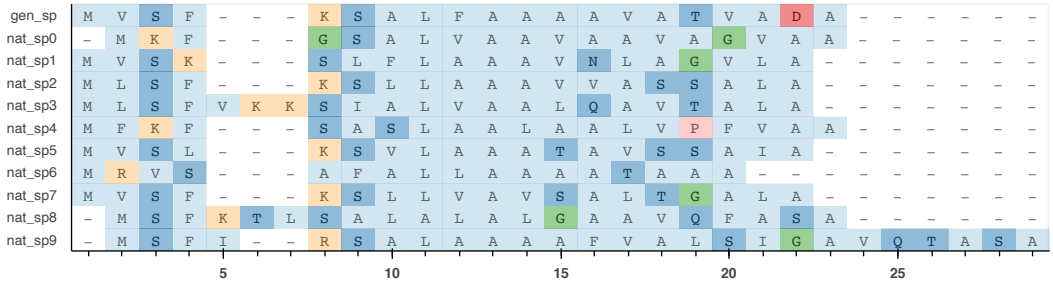


Figure 4.60: MSA for MVSFKSALFAAAAVATVADA

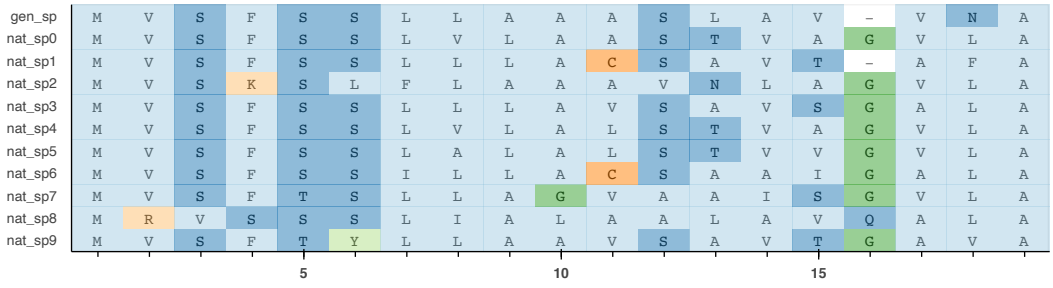


Figure 4.61: MSA for MVSFSSLLAAASLAVVNA

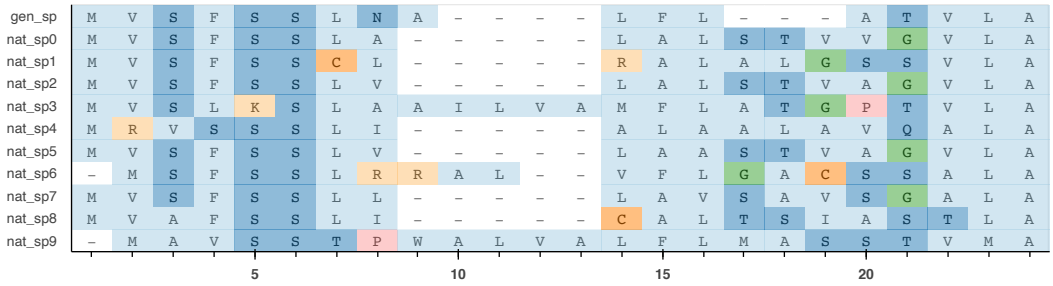


Figure 4.62: MSA for MVSFSSLNALFLATVLA

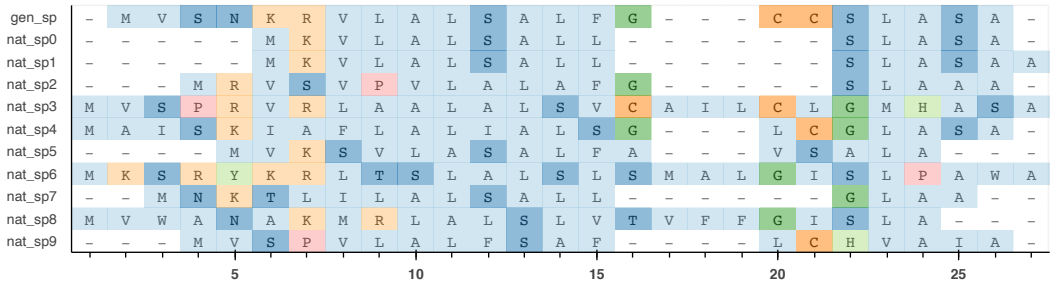


Figure 4.63: MSA for MVSNKRVLALSALFGCCSLASA

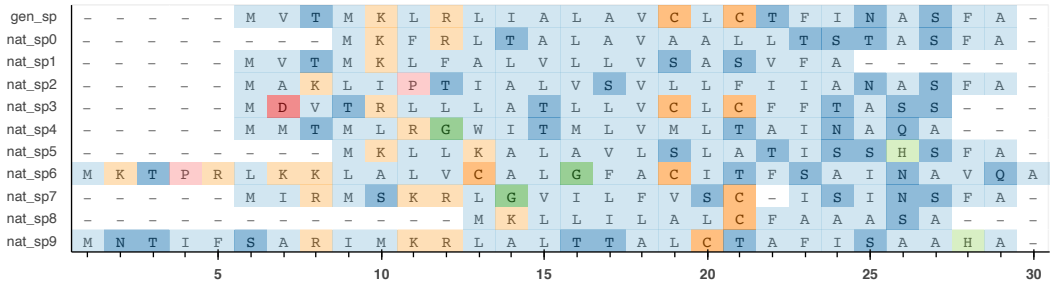


Figure 4.64: MSA for MVTMKLRLIALAVCLCTFINASFA

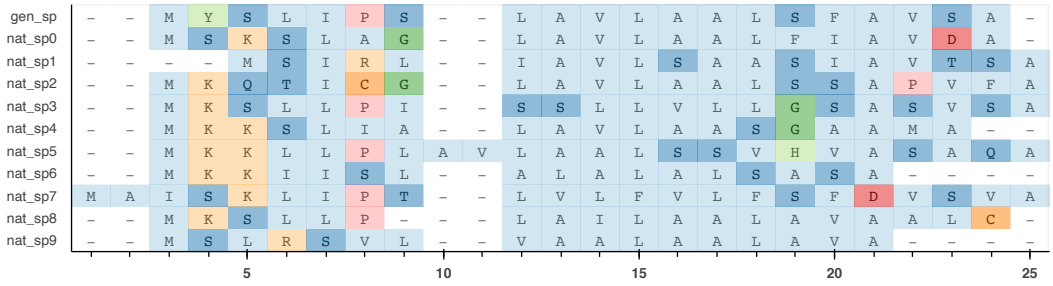


Figure 4.65: MSA for MYSLIPSLAVLAALSFAVSA

VII: Test enzyme sequences and generated SPs

The sequences of both the (input) test enzymes and the (output) Transformer generated SPs are provided below.

Table 4.11: Enzyme sequences used for testing

protein id	protein sequence (truncated)
Amy_01	ANLNGTLMQYFEWYMPNDGQHWKRLQNDSAYLAEHGITAVWIPPAYKGTSQADVGYGAYDLYDLGEFHQKGTVRTKYGTK
Amy_02	DGLNGTMMQYYEWHLENDGQHWNRLHDDAAALSDAGITAIWIPPAYKGNSQADVGYGAYDLYDLGEFNQKGTVRTKYGTK
Amy_03	ASLNGTLMQYFEWYMPNDGQHWKRLQNDSAYLAEHGITAVWIPPAYKGTSQDDVGYGAYDLYDLGEFHQKGTVRTKYGTK
Amy_04	ETANKSNELTAPSIKSGTILHAWNWSFNTLKLNMKDIHDAGYTAIQTSPINQVKEGNQGDKSMSNWYWLYQPTSYQIGNR
Amy_05	ETANKSNELTAPSIKSGTILHAWNWSFNTLKHNMKDIHDAGYTAIQTSPINQVKEGNQGNKSMSNWYWLYQPTSYQIGNR
Amy_06	ADERELKDELIYDVLVDRYFNKKIDNDYEVNALDPASFNGGDFDGMASELLFVKEMGFTALSIGPVFSTATYDGKRVLDY
Amy_07	EKEYELKDELIYDVLVDRYFNKKIDNDYEVNALDPTTFNGGDFDGMASELLFVKDMGFTALSIGSVFSTETYDGKKVLDY
Amy_08	EEKREWRDEVIYSIMIDRFNNGESKNDKQLDVGNLEGYQGGDIRGIIKRLDYIKEMGFTTIMLSPLFESEKYDGLSVRNF
Amy_09	ADWQDDAIYYVMVDRFYNGNTQNDQEVNIDDMNTYQGGDFAGITEKLDYIKEMGFTAIALNPVIQNMNGDYTGGSPHDFT
Amy_10	KEETPEIQNESIYDVLIDRFFDKNVENDYNINAKDPKAFHGGDFDGIATKLSYFQDLGFTMLSLGSVFSSETYDGQAVVD
Amy_11	AAPFNGTMMQYFEWYLSCDPQSFSEPPQCEVKVANLGITALWLPPAYKGTSRSDVGYGVYDLYDLGEFNQKGAVRTKYGT
Amy_12	AAPFNGTMMQYFEWYLPDDGTLWTKVANEANNLSSLGITALWLPPAYKGTSRSDVGYGVYDLYDLGEFNQKGTVRTKYGT
Amy_13	AQDFRARAPEDEVIYFVLPDRFENGDKANDRGGMTGDRLAHGFDPTAKGFYHGGDLKGLMARLPYIQSLGATAIWVGPIF
Amy_14	VDGKSMNPGYKTYLMAPLKKVTDYTTWEAFENDLRKAKQNGFYAVTVDFWWGDMEKNGDQQFDFSYAQRFAQAARNAGIK
Amy_15	HHNGTNGTMMQYFEWHLPNDGNHWNRLRDDAANLKSKGITAVWIPPAWKGTSQNDVGYGAYDLYDLGEFNQKGTVRTKYG
Dhg_01	SPNPTNIHTGKTLRLLYHPASQPCRSAHQFMYEIDVPFEEEVVDISTDITERQEFRDKYNPTGQVPILVDGEFTVWESVA
Dhg_02	TELILDFNKVQMRSQQLAPGVYAHLPADSAELNAKGGVAGTSGGLIVGTRGAMLIETMLNRRLFDQVQALAKKEALGLPL
Dhg_03	TYTEIVTGSTPDDRFDNLAGYPSAPHYVDVTAGDTGPLRMHYVDEGPRDGTPVVLLHGEPTWSYLYRTMIPPLAAGGCRV
Dhg_04	AGNITPTLANWSTWETFKAAQQEIVLPAGRVVHIAVTIRYVDIGEPRWGTILLMHGIPTWGCLYHAIIPPLAQAGYRVIA
Dhg_05	DVLRTPDERFEGLADWSFAPHYTEVTDADGTALRIHHVDEGPKDQRPILLMHGEPSWAYLYRKVIAELVAKGHRVVAPDL
Dhg_06	SEIGTGFPFDPHYVEVLGERMHYVDVGPRDGTPVLFLHGNPTSSYLWRNIIPHVAPSHRCIAPDLIGMGKSDKPDLDYFF
Lip_01	ENMSTTYPIVLVHGLSGFDDIVGYPYFYGIRDALEKDGHKVFTASLSAFNSNEVRGEQLWEFVQKVLKETKAKKVNLIGH
Lip_02	ATSRANDAPIVLLHGFTGWGREEMFGFKYWGGVRGDIEQWLNDNGYRTYTLAVGPLSSNWDRACEAYAQLVGGTVDYGAA
Lip_03	AEHNPVVMVHGIGGASFNFAGIKSYLVSQGWSRDKLYAVDFWDKTGTNYNNGPVLSRFVQKVLDETGAKKVDIVAHSMGG
Lip_04	ESVHNPVVLVHGISGASYNFFAIKNYLISQGWQSNKLYAIDFYDKTGNNLNNGPQLASYVDRVLKETGAKKVDIVAHSMG
Lip_05	SPIRREVSQDLFNQFNLFAQYSAAAYCGKNNDAPAGTNITCTGNACPEVEKADATFLYSFEDSGVGDVTGFLALDNTNKL
Lip_06	GLFGSTGYTKTKYPIVLTHGMLGFDSILGVDYWYGIPSSLRSDGASVYITEVSQLNTSELRGEELLDQVEEIAAISGKGK
Lip_07	ATGSGYTATKYPIVLAHGMLGFDSLLGIDYWYGIPSALRRDGAQVYVTEVSQLNTSELRGEELLAQVEEIVAISGKPKVN
Lip_08	SPVRREVSQDLFDQFNLFAQYSAAAYCAKNNDAPAGANVTCRGSICPEVEKADATFLYSFEDSGVGDVTGFLALDNTNRL
Lip_09	TTAMPSDAAIVALCAQIYQPSAPDAFEYFDAGTDDGICWAIKRLNGLDVVVLRGSRTLQDWLRDIHAFPAPSRIGHVHSG
Lip_10	AFDALFARDVALPLAAAAYSVLGGSPAVLPTGFVKTALIRADGAALTAMTDPHPAVTAMTKDTDLFGLLGHNPASRTAFV
Pro_01	AEEKVKYLIGFEEEAELEAFTEEIDQVGVFSVEEQSVAEDTLDIDVDIIDEYDYTDVLAVELDPEDVDALSEEAGISFIE
Pro_02	AGKSSTEKKYIVGFKQTMSAMSSAKKKDVISEKGGKVQKQFKYVNAAAATLDEKAVKELKKDPSVAYVEEDHIAHEYAQS
Pro_03	AGKSNGEKKYIVGFKQTMSTMSAAKKKDVISEKGGKVQKQFKYVDAASATLNEKAVKELKKDPSVAYVEEDHVAHAYAQS
Pro_04	DSASAAQPAKNVEKDYIVGFKSGVKTASVKKDIIKESGGKVDKQFRIINAAKAKLDKEALKEAKNDPDVAYVEEDHVAHA
Pro_05	AEEAKEKYLIGFNEQEAVSEFVEQVEANDEVAILSEEEEVEIELLHEFETIPVLSVELSPEDVDALELDPAISYIEEDAE
Xyl_01	ASTDYWQNWTDGGGIVNAVNGSGGNYSVNWSNTGNFVVGKGWTTGSPFRTINYNAGVWAPNGNGYLTLYGWTRSPLIEYY
Xyl_02	SPVDIDSRQASVSIDAKFKAHGKKYLGTIGDQYTLTKNSKNPAIIKADFGQLTPENSMKWDATEPNRGQFSFSGSDYLVN
Xyl_03	RTITNNEMGNHSGYDYELWKDYGNTSMTLNNGGAFSAGWNNIGNALFRKGKKFDSTRTHHQLGNISINYNASFNPGGNSY
Xyl_04	ATTITSNQTGTQDGYDYELWKDSGNTSMTLNSGGAFSAQWSNIGNALFRKGKKFDSTKTHSQLGNISINYNATFNPGGNS
Xyl_05	FPSGLTQHATGDLSKRQSITTSQTGTNNGYYYSFWTNGGGEVTYTNGDNGEYSVTWVNCGDFTSGKGWNPANAQTVTYSG

Table 4.12: Sequences of Generated SPs

sp_id	sequence	sp_id	sequence	sp_id	sequence	sp_id	sequence
sps1-1	MRFFGIHLALALATTSFA	sps11-2	MNKTIVLAASLLGLFSSTALA	sps21-3	MLKRFVKLAVIALAFAYVSA	sps31-4	MKFLILLITLGAIAATALA
sps1-2	MRQLFTSLLALLGVCSLA	sps11-3	MLKLILALCFSLPFAALA	sps21-4	MKKTGFIGKTLALVIAAGMAGTAAFA	sps32-1	MRVTSKVILTLIAATAFATAFTWSA
sps1-3	MKLSLKSIILLPTVAT	sps11-4	MKFTQAVLSLLGSAATALA	sps22-1	MKLGKLLASVAATLGVSGVNA	sps32-2	MKKFKRTILSGLALAMSIAQA
sps1-4	MKKPLGKIVASTALLISVAFSSSIASA	sps12-1	MGFRLKALLVGCLIFLAVSSAIA	sps22-2	MKKLLILACLLISSLES	sps32-3	MLFKSVLLALASAGVAVNA
sps2-1	MVATPFYLFLPWGVVAALVRSQA	sps12-2	MTSYEFLLVILGVLLSGA	sps22-3	MTKFLLSLIFITIASALA	sps32-4	MKLFKILTACLFIGLLNVSA
sps2-2	MKFFNPFKVIALACISGALATAQA	sps12-3	MPMTLLVLSLLATLFGSWVA	sps22-4	MKKTILALALLGSLAA	sps33-1	MAVMRFFASLPRRVA
sps2-3	MKKVLLATAAATLSGLMAAHA	sps12-4	MNIRLGALLAGLLLSAMASAVFA	sps23-1	MRSLGFTFLISALFGVSLSA	sps33-2	MLKRAAFLVGVSLAVAAGCGPAQA
sps2-4	MIKKIPLKTIAVMALSGCTFFVNG	sps13-1	MKNLLFSTLTAVLITSVSFA	sps23-2	MKPACRLISLLMLAVSGIASA	sps33-3	MTHRTFAALPAAALAAVSSAAFA
sps3-1	MRLIVFLATSATSLFASLA	sps13-2	MKKFAVICGLLFACIVDA	sps23-3	MMLTFFISLLFLSSALA	sps33-4	MKLSQSLTYLAVLGLAAGANA
sps3-2	MFKLKDILIGLTGILLSSLFA	sps13-3	MNKKFKTIMALAIATLSAAGVGVAHA	sps23-4	MTLKTTITLFFAALSANAAFA	sps34-1	MASKLAFFLALALAAAA
sps3-3	MLHVALLLIIGTTCSSIVSA	sps13-4	MKKSLISFLALGLLFGSAFA	sps24-1	MRAKALAASLAGALAGAASA	sps34-2	MKFLSLKLVVLAFYVAFQINA
sps3-4	MRLAKIAGLTASLLFSLWGALA	sps14-1	MALANKFFLLVALGLSVSG	sps24-2	MVSLSFSLVASAVTHVASA	sps34-3	MAKLIALVLLGLAAA
sps4-1	MVGYSTAWLLLLAASVIASG	sps14-2	MVIVLTSIILALWNAQA	sps24-3	MVSFSSLNALFLATVLA	sps34-4	MRSLLLTLLGALLRA
sps4-2	MAVNTKLIGVSLYSFTPFLVFA	sps14-3	MTKFLLSLAVLATAVASA	sps24-4	MKFQDLTLVLSLSTALA	sps35-1	MKLNIVKLLVLAAFAQAASA
sps4-3	MLGRGALTAAILAGVATADS	sps14-4	MKFLSIVLLIVGLAYG	sps25-1	MRVLSATAFLALLAFGLSGATA	sps35-2	MILFYVLPVVLALVSG
sps4-4	MAILVLLFLLAVEINS	sps15-1	MMAAVVRAVAATLILILCGAELA	sps25-2	MKFLSTAFVLLIALVAGCSTA	sps35-3	MKKNLLKLTLALISGMSQFA
sps5-1	MLLPAFMLLILPAALA	sps15-2	MLPTAAFLSVNLLLTGAFFGCA	sps25-3	MLKRFLTLFLGFLALASSLA	sps35-4	MKFLIPLFVLFIVFGNAYA
sps5-2	MKMRTGKKGFLSILLAFLLVITSIPFTLVDVEA	sps15-3	MYSLIPSLAVLAALSFAVSA	sps25-4	MKLLTSFVLIGALAFA	sps36-1	MKRVFSLFTAVCGLLSVSA
sps5-3	MSNKPAKCLAVLAAIATLSATQA	sps15-4	MFKFVLVLSVLAALASARA	sps26-1	MLKKLAMAVGAMLTSISFLLPSSAQA	sps36-2	MKKFSIFLVLSITVLA
sps5-4	MKMRTGKKGFLSILLAFLLVITSIPFTLVDVEA	sps16-1	MRVPYLIASLLALAVSLFSTATA	sps26-2	MKKLLVIAALACGVATAQA	sps36-3	MKKKIVAVLTLSVVLA
sps6-1	MKLGFLTSFVAYLTSAA	sps16-2	MKKIKSILVLALIGIMSSALA	sps26-3	MIKTLLVSSILIPCLATGA	sps36-4	MKKRVISALAALWLSVLGAPAVLA
sps6-2	MKLSTIFVRFLAIALLATMSTAQA	sps16-3	MLGAKFLWTVLFSLSLSLAHA	sps26-4	MGIQKKVSILVAGLFMATAFATA	sps37-1	MGVFSFLTTEAMAVFLAGLAHA
sps6-3	MQRSLFLVLSLVSSVASA	sps16-4	MLTFHRIIRKGWMFLLAFLLTALLFCPTGQPAKA	sps27-1	MKKIVALFLVFCFLAG	sps37-2	MTMKGLRVVALVVLASLGIFA
sps6-4	MKLFTATIAVLGAVSATAHA	sps17-1	MLIRKYLSFAISLLIATALPASA	sps27-2	MNKKVLAAIVLGMLSVFTSAAQA	sps37-3	MTKFLSASLALLSGLATASSDA
sps7-1	MLKNFLLASLAICVTFSATG	sps17-2	MEKVLLRLLILLSLLAGALSFA	sps27-3	MKKTAIASALLALPFVFA	sps37-4	MTQKSLLLALTAVALVSVNA
sps7-2	MVKNFQKILVLALLIVCCSSISLATFA	sps17-3	MKLGSIFLFALFLACSAEA	sps27-4	MKKTAAIAALAGLSFAGMAHA	sps38-1	MNRLYAVFAVLCFAQVLHG
sps7-3	MKLLPAFFLITAATVASA	sps17-4	MNLKILFALALGVCLAA	sps28-1	MISANKILFLILCVACVSA	sps38-2	MKKLLLQSLILSELGGCLA
sps7-4	MKDLFRLIALLSCCLALFPLTFA	sps18-1	MTRPAPAFRLSLVILCLAIPAADA	sps28-2	MVKLASILLIILAGESFA	sps38-3	MAARSVLLLALLTLAVSTA
sps8-1	MRKTAVSFTVCALMLGTAMA	sps18-2	MVTMKLRLIALAVCLCTFINASFA	sps28-3	MINKLIALTVLFSLGINA	sps38-4	MKGTLAFLLVFLLNLYVHG
sps8-2	MKKFCKILVISMLAVLGLTPAAVA	sps18-3	MTKLLAVIAASLMFAASTFA	sps28-4	MVASLWSSILPVLAFLWADLSAGA	sps39-1	MLSIDTSSTRRVVPNTALFPNTHRRDFATAGQLLAMASAVLTGAPAHA
sps8-3	MKKSLSAVLLGVALSAVASSAFA	sps18-4	MVSNKRVLALSALFGCCSLASA	sps29-1	MKFLLFIALSLAVATAA	sps39-2	MNISIFVGKLALAALGSALVA
sps8-4	MKSLLLTAFAAGTALA	sps19-1	MVSFKSALFAAAAVATVADA	sps29-2	MRHFLSLLLYGATLVSSSACS	sps39-3	MRRLFLLSSLASLSVASA
sps9-1	MLSLKSLFLSTLLIVLAASGFA	sps19-2	MQKKTAIAIAAGTAIATVAAGTQA	sps29-3	MKFSAIVLLAALAFAVSA	sps39-4	MKCCRIMFVLLGLWFVFGLSVPGGRTEA
sps9-2	MKKRLHIGLLLSLIAFQAGFA	sps19-3	MVSFSSLLAAASLAVVNA	sps29-4	MKKRLLIASVALGSLFSFCA	sps40-1	MKFLILATLSIFTGILA
sps9-3	MKLLAFIFALFLFSIARA	sps19-4	MKNFATLSAVLAGATALA	sps30-1	MSWRSIFLLVLLASIDFING	sps40-2	MKVFTLAFFLAIIVSQA
sps9-4	MNKLFYLFMLGLAAFA	sps20-1	MKLNKLLSIAAGCTVLGSTYALA	sps30-2	MRLPSLLLPLAALIA	sps40-3	MKKKIAITLLFLSLLNRA
sps10-1	MKFSTILAAAILVGVRA	sps20-2	MKLKKLGVILAICLGISSTFA	sps30-3	MKVLAALVLALVATASA	sps40-4	MKLLKVIATAFLGLTSFASA
sps10-2	MKVFTLAFAIICQLFASA	sps20-3	MKKLLLAACVLFSLASVSA	sps30-4	MARA	sps41-1	MPTVVALDLATYVLQPSKRA
sps10-3	MKKKIAIILMSLLLNTIASTFA	sps20-4	MIRLKRLLAGLLLPLFVTAFG	sps31-1	MRKLLIWLAGFLVLILKT	sps41-2	MLMVPLLLALGAVAAG
sps10-4	MKLKIVFAVAAIAPVLHS	sps21-1	MTRSLFIFSLLALAIFSGVSASA	sps31-2	MRKFISSLLLGLVVSIATAVA	sps41-3	MPAARRLALFAAVALAAVGLSPAALA
sps11-1	MVYTSILLAASAATVQA	sps21-2	MKLIPNKKTLIAGILAISTSFAYS	sps31-3	MNTLFLFTSLFLFLFAKVTA	sps41-4	MRSLLLTSALAALVSLAAASA

4.7 Bibliography

References

1. Tsirigotaki, A., De Geyter, J., Šoštaric, N., Economou, A. & Karamanou, S. Protein export through the bacterial Sec pathway. *Nature Reviews Microbiology* **15,** 21 (2017).

2. Low, K. O., Mahadi, N. M. & Illias, R. M. Optimisation of signal peptide for recombinant protein secretion in bacterial hosts. *Applied Microbiology and Biotechnology* **97,** 3811–3826. doi:10.1007/s00253-013-4831-z (2013).

3. Driessen, A. J., Manting, E. H. & van der Does, C. The structural basis of protein targeting and translocation in bacteria. *Nature Structural Biology* **8,** 492–498 (2001).

4. Osborne, A. R., Rapoport, T. A. & van den Berg, B. Protein translocation by the Sec61/SecY channel. *Annual Review of Cell and Developmental Biology* **21,** 529–550 (2005).

5. Wickner, W., Driessen, A. J. & Haril, F.-U. The enzymology of protein translocation across the Escherichia coli plasma membrane. *Annual Review of Biochemistry* **60,** 101–124 (1991).

6. Berks, B. C., Palmer, T. & Sargent, F. Protein targeting by the bacterial twin-arginine translocation (Tat) pathway. *Current Opinion in Microbiology* **8,** 174–181 (2005).

7. Natale, P., Brüser, T. & Driessen, A. J. Sec-and Tat-mediated protein secretion across the bacterial cytoplasmic membrane–distinct translocases and mechanisms. *Biochimica et Biophysica Acta (BBA)-Biomembranes* **1778,** 1735–1756 (2008).

8. Mori, A. *et al.* Signal peptide optimization tool for the secretion of recombinant protein from Saccharomyces cerevisiae. *Journal of Bioscience and Bioengineering* **120,** 518–525 (2015).

9. Zhang, L., Leng, Q. & Mixson, A. J. Alteration in the IL-2 signal peptide affects secretion of proteins in vitro and in vivo. *The Journal of Gene Medicine* **7,** 354–365 (2005).

10. Mergulhão, F., Summers, D. K. & Monteiro, G. A. Recombinant protein secretion in Escherichia coli. *Biotechnology Advances* **23,** 177–202 (2005).

11. Nielsen, H., Engelbrecht, J., Brunak, S. & von Heijne, G. Identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites. *Protein Engineering* **10,** 1–6 (1997).

12. Nielsen, H. & Krogh, A. *Prediction of signal peptides and signal anchors by a hidden Markov model.* in *ISMB Proceedings* **6** (1998), 122–130.

13. Armenteros, J. A. *et al.* Detecting Novel Sequence Signals in Targeting Peptides Using Deep Learning. *Life Science Alliance* **2.** doi:10.1101/639203 (2019).

14. Brookes, D. H., Park, H. & Listgarten, J. Conditioning by adaptive sampling for robust design. *arXiv* (2019).

15. Costello, Z. & Garcia Martin, H. How to Hallucinate Functional Proteins. *arXiv* (2019).

16. Madani, A. *et al.* ProGen: Language Modeling for Protein Generation. *arXiv* (2020).

17. Repecka, D. *et al.* Expanding functional protein sequence space using generative adversarial networks. *bioRxiv,* 789719 (2019).

18. Riesselman, A. J., Ingraham, J. B. & Marks, D. S. Deep generative models of genetic variation capture the effects of mutations. *Nature Methods* **15,** 816–822 (2018).

19. Brockmeier, U. *et al.* Systematic screening of all signal peptides from Bacillus subtilis: a powerful strategy in optimizing heterologous protein secretion in Gram-positive bacteria. *Journal of Molecular Biology* **362,** 393–402 (2006).

20. Degering, C. *et al.* Optimization of protease secretion in Bacillus subtilis and Bacillus licheniformis by screening of homologous and heterologous signal peptides. *Applied and Environmental Microbiology* **76,** 6370–6376. doi:10.1128/aem.01146-10 (2010).

21. Hemmerich, J. *et al.* Use of a Sec signal peptide library from Bacillus subtilis for the optimization of cutinase secretion in Corynebacterium glutamicum. *Microbial Cell Factories* **15,** 208 (2016).

22. Engelman, D. & Steitz, T. The spontaneous insertion of proteins into and across membranes: the helical hairpin hypothesis. *Cell* **23,** 411–422 (1981).

23. Duffy, J., Patham, B. & Mensa-Wilmot, K. Discovery of functional motifs in h-regions of trypanosome signal sequences. *Biochemical Journal* **426,** 135–145 (2010).

24. Freudl, R. Signal peptides for recombinant protein secretion in bacterial expression systems. *Microbial Cell Factories* **17,** 52 (2018).

25. Owji, H., Nezafat, N., Negahdaripour, M., Hajiebrahimi, A. & Ghasemi, Y. A comprehensive review of signal peptides: Structure, roles, and applications. *European Journal of Cell Biology* **97,** 422–441 (2018).

26. Vaswani, A. *et al. Attention is all you need* in *Advances in Neural Information Processing Systems* (2017), 5998–6008.

27. Consortium, U. UniProt: a worldwide hub of protein knowledge. *Nucleic Acids Research* **47,** D506–d515 (2019).

28. Bahdanau, D., Cho, K. & Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* (2014).

29. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* (2018).

30. Liu, Y. *et al.* Roberta: A robustly optimized bert pretraining approach. *arXiv* (2019).

31. Rao, R. *et al. Evaluating protein transfer learning with TAPE* in *Advances in Neural Information Processing Systems* (2019), 9686–9698.

32. Rives, A. *et al.* Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv,* 622803 (2019).

33. Graves, A. Sequence transduction with recurrent neural networks. *arXiv* (2012).

34. Eddy, S. R. Profile hidden Markov models. *Bioinformatics* **14,** 755–763 (1998).

35. Kingma, D. P. & Welling, M. Auto-encoding variational Bayes. *arXiv* (2013).

36. Zhang, W. *et al.* Optimal secretion of alkali-tolerant xylanase in Bacillus subtilis by signal peptide screening. *Applied Microbiology and Biotechnology* **100,** 8745–8756 (2016).

37. Bedbrook, C. N. *et al.* Machine learning-guided channelrhodopsin engineering enables minimally invasive optogenetics. *Nature Methods,* 1–9 (2019).

38. Thompson, J. D., Higgins, D. G. & Gibson, T. J. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research* **22,** 4673–4680 (1994).

39. Cock, P. J. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25,** 1422–1423 (2009).

40. Grote, A. *et al.* JCat: a novel tool to adapt codon usage of a target gene to its potential expression host. *Nucleic Acids Research* **33,** W526–w531 (2005).

41. Koo, B.-M. *et al.* Construction and analysis of two genome-scale deletion libraries for Bacillus subtilis. *Cell Systems* **4,** 291–305 (2017).

42. Madeira, F. *et al.* The EMBL-EBI search and sequence analysis tools APIs in 2019. *Nucleic Acids Research* **47,** W636–w641 (2019).

43. Armenteros, J. J. A. *et al.* SignalP 5.0 improves signal peptide predictions using deep neural networks. *Nature Biotechnology* **37,** 420–423 (2019).

Chapter 5

LANGUAGE MODELING FOR ENZYME-SUBSTRATE INTERACTIONS

Contributions Statement: Z.W. conceived of the idea. A.F., A.S., L.P., A.S., and V.N. participated in its implementation. Z.W. identified and cleaned relevant datasets for training and validation.

5.1 Abstract

While directed evolution has been remarkably successful in optimizing protein sequences, it requires a starting point with measurable function. In this chapter, we begin to address the issue of modeling the interactions between enzymes and their substrates from a language modeling perspective. While we were able to predict whether an enzyme has activity on a new substrate with some success, this approach was not able to generate convincing enzyme sequences given a substrate. In the future, incorporating data from alternative sources, including perhaps structural information, is likely necessary for this approach to succeed.

5.2 Introduction

Enzymes have been engineered by nature to accomplish the reactions necessary for life. To adapt these protein catalysts to human tasks, directed evolution [1] has emerged as a simple yet powerful tool that mirrors natural optimization with high success. However, directed evolution of functional enzymes is reliant on the identification of a suitable starting point with measurable starting activity. While domain experts supplemented by serendipity have developed the intuition necessary for identifying this parent protein for specific reactions, engineering enzymes with new functionality from scratch remains a fundamental challenge despite its importance. For example, predicting new substrates for a given protein has potential applications in developing drug discovery panels [2], understanding enzymatic promiscuity [3], designing metabolic pathways [4], and combating antibiotic resistance [5].

In this chapter, we begin to tackle the task of modeling the interactions between an enzyme and its substrate and attempt to predict whether a new enzyme-substrate pair will have activity. Specifically, we train a deep generative model for enzyme-substrate pairs to simultaneously predict possible mutations in a given enzyme, alterations to

functional groups in a given substrate, and an interaction probability between the pair that assesses how likely they are to interact. To compare these heterogeneous data types, we use a three-component model that is jointly combined with multi-headed self-attention:

1. **Protein representation:** We represent enzymes by their primary sequence, learning an embedding for each of the 25 amino acids (including noncanonical amino acids) present in BRENDA.

2. **Substrate representation:** We represent small molecule substrates as a connection graph formed by a tree decomposition that avoids cycles, as implemented by Jin *et al* [6] and inspired by earlier work in chemistry for quantifying molecular similarity [7].

3. **Interaction flag:** We incorporate a binary variable describing whether the enzyme and substrate are known to interact. Negative examples are generated by randomly shuffling enzyme-substrate pairs in the input.

We train this model using self-supervised input recovery tasks, as analogous to various natural language representation tasks [8, 9] and protein sequence representation tasks [10–13] coupled to a multi-task loss function. We demonstrate that this learned representation has some success in predicting whether an enzyme has activity for a given substrate on two tasks. However, we discover that the trained model has not learned a meaningful representation of protein sequence *position* when examining the model more closely. While potential engineering improvements to training this model exist [14, 15], we suspect that there is an insufficient amount of high quality data to model the dynamic interactions between diverse enzyme families and their substrates from sequence representations alone.

5.3 Background and Related Work

Enzymes are biologically encoded and synthesized as a linear chain of amino acids, of which there are 20 canonical members. This linear chain then folds into a dynamic 3D structure to accomplish its biological purpose. There are two engineering paradigms for adapting these sequences for human purposes: rational design and directed evolution.

Rational design has recently emerged with tremendous success for proteins [16] by computing the structure of designed proteins. However, small errors in energy

calculation can have critical effects in protein design [17]. This is particularly true for enzymes, where small errors in atomic distances have large effects in corresponding bond energies [18].

Alternatively, directed evolution has also enjoyed quite some success [19], but requires a starting point with measurable activity. While trained protein engineers have been able to identify potential starting points through deep mechanistic understanding and high-throughput screening, there is a large enzymatic database with highly relevant information that has yet to be fully leveraged called BRENDA [20]. We develop a sequence-based model to learn a representation of enzymatic systems based on the protein-substrate data that can be extracted from BRENDA.

An analog in deep learning research to our sequence modeling problem can be found in natural language processing (NLP). Recently, the attention mechanism, which models dependencies regardless of sequential distance, has become a dominating force in language modeling [8, 9, 21]. We adapt the Transformer encoder to our protein-substrate representation, forming a joint attention model to learn from both components of the enzymatic system. By learning which positions in the sequence to attend to, we hypothesized that the attention mechanism would accurately recapitulate critical 3D dependencies in the physical enzymatic system.

Enzyme-substrate interactions have been modeled previously in efforts to predict their Enzyme Commission (EC) number, the dominant enzyme classification scheme established in 1961 that contains 4 layers of specificity in a classification tree [22]. There are currently two main strategies for classifying enzymes by EC number. One approach is to focus on the substrate, and use the molecular similarity of enzyme substrates to classify the enzyme [23]. An alternate strategy is to focus on the protein sequence, and to use deep learning techniques to predict the EC number [24–27]. We develop a representation that directly represents both components of an enzymatic system, and eschew the EC number classification system in hopes of uncovering physically meaningful intermediate representations.

Other machine learning methods have been applied to various protein engineering tasks, which is better reviewed elsewhere [28]. However, recently a number of exciting deep learning approaches have been applied to understanding protein sequences. Self-supervised pre-training have been particularly useful across a range of protein classification and annotation tasks [10, 11, 27, 29]. Their success attests to the ability of deep learning models to model the exponentially-growing amounts of unlabeled biological data being generated.

In small molecule modeling, Deep learning has had similar success in drug target identification [30], materials discovery [31], and synthesis applications [32], which are better reviewed elsewhere [33]. While small molecules can be represented in the sequential SMILES format [34], this notation was developed to convert 3-dimensional molecules to ASCII form and was not designed to capture molecular similarity. To this end, nearly identical molecules can have markedly different strings, as cyclical substructures within molecules do not lend themselves to a linear ASCII representation. Inspired by recent work [6, 33, 35, 36], we instead translate molecules using deterministic mappings from RDKit [37].

To our knowledge, this work is among the first to directly embed an enzyme-substrate representation for deep learning. While studies exists in protein-drug *binding* that are similar in spirit [36, 38], these approaches typically separate the protein and target encodings and, more importantly, are developed for protein binding, a simpler modeling task as only a single optimal energy state must be modeled.

5.4 Training Data Collection and Representation

We obtained enzyme reaction information from the BRENDA database [20]. BRENDA categorizes reactions by EC number (example: EC 1.1.1.1), where each BRENDA record contains references to member proteins and reaction substrates and products. Within each BRENDA record, each protein is matched to each substrate by protein identifiers unique to that record. While some proteins are explicitly matched to substrates by protein identifiers, we elect to match all proteins to all substrates within each BRENDA record by their respective identifiers to generate sequence-substrate pairs.

Our data processing pipeline converts BRENDA records into collections of sequence-substrate pairs via the Uniprot protein sequence database and substrate-smiles databases. To resolve protein sequences, we extract Uniprot accession keys when possible from within the BRENDA protein identifier and pair this accession key to the Uniprot90 cluster and sequence from the Uniprot database. For substrates, we extract common names from substrate lines and then convert to SMILES strings via a collection of available dictionaries. If the reaction line contains the {r} reversibility tag, we also extract the labeled products as substrates; we do not assume all reactions are reversible.

We processed the BRENDA 2019.1 (January) release and extracted 671,438 individual protein-substrate functional pairs with this pipeline. We selected and withheld 251

random UniRef90 clusters and 99 protein-substrate pairs from downstream validation tasks as an out-of-sample test partition (200130_test) containing 6313 test records. All remaining entries were included as a training set (200130_train) containing 665,125 training records. This training set was randomly subdivided into 80/10/10 in-sample train/test/validation partitions.

If the protein-substrate pair does not already exist as a functional pair, we specify this as nonfunctional to augment the dataset of functional protein-substrate pairs, assuming that there is a low probability that a random combination of protein-substrate is unreactive. With 27322 unique protein sequences and 15059 unique molecules, there is a total of over 400 million possible pairs. We select a 1:1 ratio of functional protein-substrate pairs and randomly-generated nonfunctional protein-substrate pairs to be generated during training.

The data representation cused is a concatenation of 1) the protein sequence as amino acids, 2) molecular tree (detailed below), and 3) a binary functionality token (<TRUE> or <FALSE>) for whether a protein-substrate pair exists in BRENDA. The concatenated representation has 1281 tokens, with 1024 amino acid characters for the protein, 45 molecular tree characters for the substrate, and 1 binary token specifying whether the enzyme has activity on the substrate.

The molecular tree representation is adapted from a tree decomposition implemented by Jin *et al* [35], as shown in 5.1. After the tree decomposition, the approach diverges from that of Jin, who uses message passing networks to embed the molecular tree. We instead directly encode the molecular adjacency graph as a 2D connection matrix with 1D embeddings for each node. 569 unique nodes were identified from the January 2019 BRENDA release. These 1D embeddings are then tiled for each node in both vertical and horizontal directions, to form two 2D representations for each node. These 2D representations are then stacked with the 2D connection matrix, and the stacked channels are upsampled to the hidden dimension of the attention sequence, while the 2D representation is convolved down to a 1D with a series of (3x1), (3x1), and (5x1) convolutions. Originally, we attempted to use the embedded SMILES string representation of a molecule. However, we were did not observe a decrease in loss during training, and proceeded with this graphical representation.

5.5 Training Task and Model

Self-supervision and multitask loss: To be useful for enzymatic systems, a predictive model must capture the interactions between protein mutations and

molecular substrate changes. To form our self-supervised training procedure, we randomly remove 15% of the input enzyme and replace those tokens with a MASK token. At the start of training, an equal amount of data for nonfunctional protein-substrate pairs is generated by randomly pairing proteins and substrates present in BRENDA. Any true functional pairs generated are passed over. These records are assigned a probability of 0.001 in a form of label smoothing.

We use a four-part loss function that uses cross-entropy to fit the sequence recovery task on the enzyme L_{prot}, the molecular adjacency matrix L_{mat}, the molecular node identity recovery task on the substrate L_{nodes}, and the prediction of substrate-enzyme functionality L_{func}. The final total loss is a weighted summation of the individual cross entropy losses. With the exception of L_{func}, each individual loss component is roughly scaled to the starting training loss of the protein loss by scaling L_{mat} up by 10-fold. Due to the importance of L_{func} in specifying interactions, this loss was initially scaled to be approximately 2 times that of the protein training loss. The scaling factor for L_{mat} and L_{nodes} is scaled every 10 epochs to contribute equally to L_{prot}, while L_{func} is scaled to contribute twice that of L_{prot}. In other words, every 10 epochs, scaling factors σ are set such that:

$$\sigma_{mat} \times L_{mat} = L_{prot}$$

$$\sigma_{nodes} \times L_{nodes} = L_{prot}$$

$$\sigma_{func} \times L_{func} = 2L_{prot}$$

Architecture: Our model architecture is described in Fig.5.1. To summarize, each component of the enzymatic system (the enzyme, substrate, and flag) is embedded in some manner with the identical number of hidden dimensions. The exact embeddings are described previously in the Data Collection and Representation section. These are then passed through $N = 6$ layers of self-attention as implemented in the Transformer encoder [8], and a final linear layer to make the final token predictions for each respective portion.

Model Training: We train our model on 1 V100 GPU, with batch size of 14. We use the Adam optimizer, with linear warm-up over 1×10^1 steps to a learning rate of 4×10^{-5}. The learning rate then proceeds through a cyclical cosine annealing step with cycles of 4.2×10^5 steps and a final value of 4×10^{-7}.

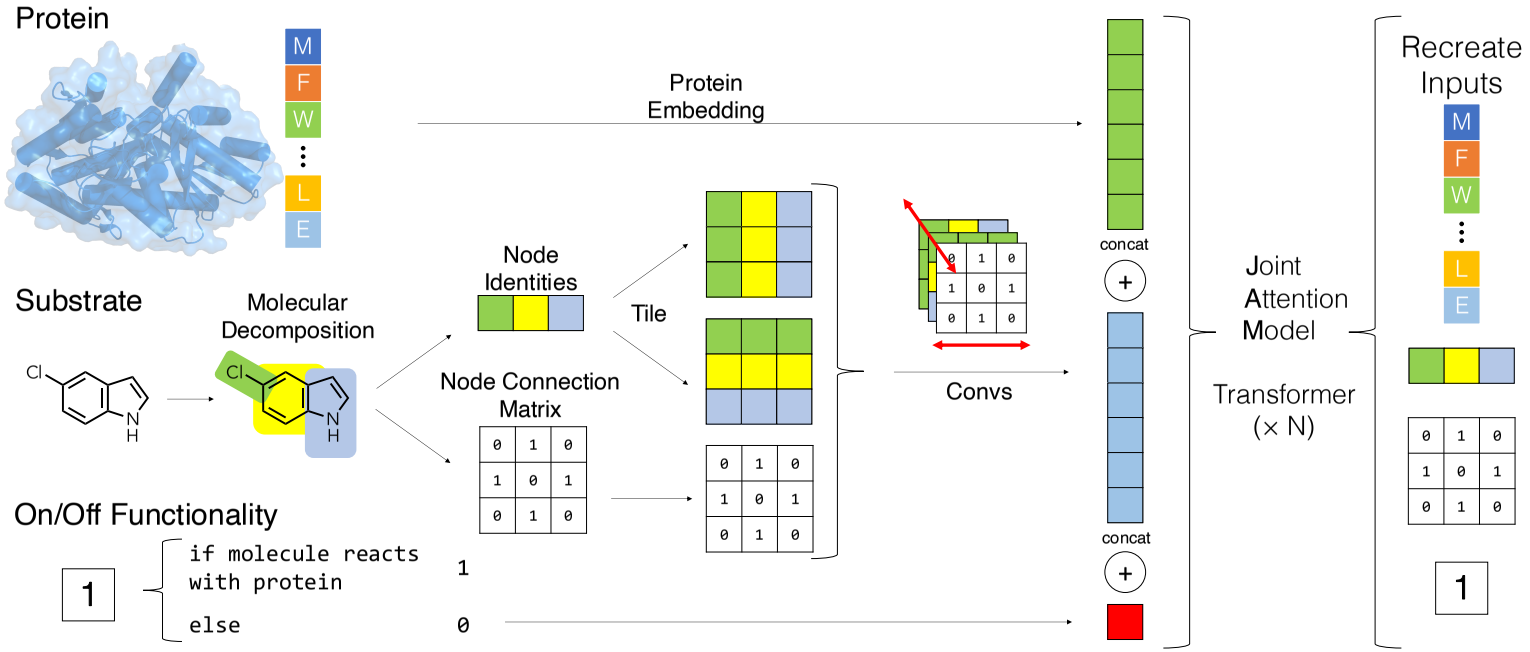


Figure 5.1: The architecture diagram for our enzyme-substrate model. We use three heterogeneous data sources: the protein primary sequence, the substrate, and the functionality relationship flag. We directly embed the 21 unique amino acids in the protein primary sequence, as well as the binary function flag. For each substrate, we use the tree-based molecular decomposition from Jin *et al* [35] to form connection graphs of identified molecular subunits, and embed each of these 568 nodes. Because substrates have relatively few nodes, we represent their graphical structure with the binary connection matrix, with each node containing an embedding with d_{hidden} equal to the maximum size of the connection. We then tile each 1D embedding in the horizontal and vertical directions to from 2D matrices of the same shape as the connection matrix. Each position in these matrices is then convolved to a linear representation of length 45 to return an embedding of the substrate. After concatenating, we use the sinusoidal positional embedding[8], followed by $N = 6$ layers of self-attention. We have one final linear layer to predict the output.

5.6 Results

Predicting functionality of enzyme-substrate pairs

The ability to predict whether a substrate reacts with a protein is critical for both genomic annotation and directed evolution. In genomic annotation, a large amount of sequence information must be parsed in order to identify which proteins and metabolic pathways may exist. For directed evolution, a starting point with measurable activity for the target reaction must be identified. This is typically accomplished by screening large, in-house compilation plates of sequences evolved for various tasks, and can be greatly expedited with a predictive model.

BRENDA

An initial test for the unsupervised learning task is to infer the identity of the ablated functionality flag. To test the performance of our model in generating protein sequences and functional tags, we train the model on the January 2019 release of BRENDA, and ask the model to predict functional protein sequences and functionality tags new to the July 2019 release. This time-based train/test split was inspired by BridgIt [23], which uses a substrate molecular representation to model similarity between proteins, and validates through various releases of an alternate database we were unable to access.

Therefore, as a test set, 3730 new BRENDA records present in the 2019.2 release that were *not* present in the 2019.1 release were used. In examining the new enzyme-substrate pairs, we masked the identity of the functionality flag, and ask the unsupervised model to predict <TRUE> or <FALSE>. Of these 3730 new records, 2900 (78%) were correctly identified as functional protein-substrate pairs.

As BRENDA only labels <TRUE> enzyme-substrate pairs, we also randomly generate 3730 <FALSE> enzyme-substrate pairs by matching enzyme sequences to substrates that are not within the same EC classification. The pool of possible pairs is restricted to enzymes and substrates that are present in the new records added to the 2019.2 release. The resulting ROC curve is shown below in Figure 5.2 with an AUC of 0.789. For comparison, we also show the results from a model trained without considering the loss from the functionality flag.

Human cytochrome P450s

While BRENDA is a curated database for enzyme annotation, we also sought to validate our predictions on an external task. For this, we chose CypReact, which

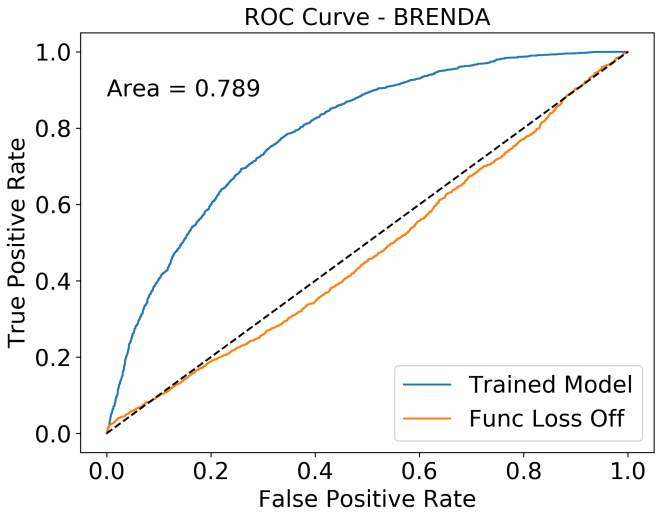


Figure 5.2: ROC curve for reactant/non-reactant classification of functional protein-substrate pairs new to BRENDA's July 2019 release compared to BRENDA January 2019. Of the 3730 explicit true protein-substrate new pairs, 2900 (78%) are correctly identified as functional.

contains data on over 1700 substrates against the 9 most common human cytochrome P450 (CYP450) isoforms [39]. Understanding CYP450 activity against diverse substrates is particularly important as P450s are a major component of Phase I drug metabolism [40, 41], and their bacterial homologs' enzymatic promiscuity has been heavily engineered for biocatalysis as well [42, 43]. To this end, the authors of CypReact train independent models with 1632 substrates for each of 9 CYP450 variants, for a total of 14688 training data. Critically, this is one of the few enzymatic databases that contains explicit non-reactant labels. For comparison, the BRENDA 2019.1 database contained 32 explicit protein-substrate pairs to the same 9 variants, from which 612 implicit records were generated by matching the protein to other substrates within the same EC classification.

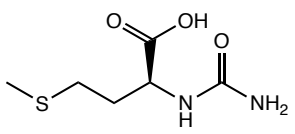
Although CypReact was trained with significantly more data in a supervised prediction model, we make the comparison with their test set in Table 5.1. We also compare our results with CypReact's implementation of a competing program, SmartCYP [44], and a commercially-developed drug metabolism prediction software ADMET Predictor [45]. EnzPred is competitive with some of the existing software despite being trained with data obtained with an entirely different focus, although CypReact remains dominant.

	1A2	2A6	2B6	2C8	2C9	2C19	2D6	2E1	3A4
Model	AUC								
CypReact	86%	84%	86%	84%	83%	83%	87%	87%	92%
SmartCyp					51%		49%		60%
ADMET	79%	77%	74%	68%	74%	75%	81%	75%	75%
EnzPred	74%	66%	73%	72%	65%	66%	75%	56%	64%
Test Data Distribution									
# Reactants	24	6	4	12	28	20	21	6	32
# Non	100	100	100	100	100	100	100	100	100

Table 5.1: Results on the functionality classification for human cytochrome P450 (CYP450) isoforms with data splits obtained from CypReact [39]. CypReact trained individual supervised models for each CYP450 isoform with 1632 substrates for a total of 14688 training data, and various features derived from molecular fingerprints and physical properties. Our EnzPred model was trained on data parsed from BRENDA v2019.1, which contained 32 explicit CYP450-substrate pairs, and 612 pairs inferred by matching substrated within each EC class to each CYP variant.

Protein Generation

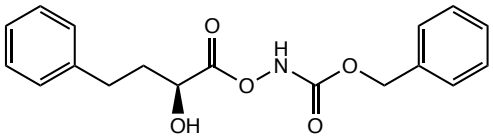
We attempted to generate protein sequences given a desired small molecule on which a desired enzymatic reaction is to occur. These molecules were converted to the molecular graph representation, and beam search [46] was used to generate protein sequences from a starting methionine, with a beam size of five. Sample substrate molecular structures, and the first 20 generated amino acids are show in Figure 5.3. Unfortunately, generated protein sequences are prone to repetition, suggesting that the models are failing to learn meaningful position variation. While the number of unique enzyme-substrate *paired* entries is high (over 600,000), we suspect that the number of individual unique enzymes (27,322) is too low for the Transformer model.



carbamoyl-*L*-methionine

CSCC[C@H](NC(N)=O)C(O)=O

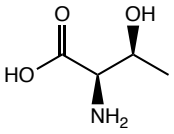
MKKKKKKKDKKKKKKKKKK



benzyl (S)-((2-hydroxy-4-phenylbutanoyl)oxy)carbamate

O[C@H(C(=O)ONC(OCC1=CC=CC=C1)=O)CCC1=CC=CC=C1

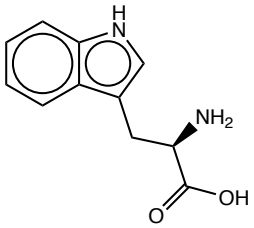
MAQQQSLLVLLLLLLLLLL



D-threonine

C[C@@H](O)[C@H](N)C(O)=O

MLLLLLLLLLLLLLSLLLL



D-tryptophan

N[C@H](Cc1c[nH]c2ccccc12)C(O)=O

MNNNKNNNNNNNNNNNNN

Figure 5.3: Sample generated amino acid sequences given SMILES strings. Sequences were generated by beam search from left to right, with a beam size of five, given a starting Methionine (M). The structures of the molecules as well as the common name are also shown. Generated protein sequences are prone to repetition, suggesting that the model has not learned positional information.

5.7 Conclusions and Future Work

In this chapter, we have developed and trained a model to jointly capture the interactions between an enzyme's amino acid sequence and its small molecule substrate. While we were able to show some capabilities in predicting whether enzymes and substrates interact, we ultimately found that this model did not capture sufficient positional information for the enzyme sequence. We suspect this is due to a low amount of available training enzymes available (27,322 sequences). In the future, combining approaches where representations are first trained on large databases of protein sequences and small molecules, such as UniRef [47] and ZINC [48] respectively, may address this issue. Additionally, incorporating more detailed structural information such as the estimated active site residues will likely improve this approach. In any case, generating functional protein sequences is a monumental task that we must continue to explore to adapt proteins to human applications.

5.8 Bibliography

References

1. Romero, P. A., Krause, A. & Arnold, F. H. Navigating the protein fitness landscape with Gaussian processes. *Proceedings of the National Academy of Sciences USA* **110,** e193–e201. doi:10.1073/pnas.1215251110 (2013).
2. Principles of early drug discovery. *British Journal of Pharmacology* **162,** 1239–1249. doi:10.1111/j.1476-5381.2010.01127.x (Mar. 2011).
3. Khersonsky, O. & Tawfik, D. S. Enzyme promiscuity: a mechanistic and evolutionary perspective. *Annual Revies of Biochemistry* **79,** 471–505. doi:10.1146/annurev-biochem-030409-143718 (2010).
4. Wang, L., Dash, S., Ng, C. Y. & Maranas, C. D. A review of computational tools for design and reconstruction of metabolic pathways. *Synthetic and Systems Biotechnology* **2,** 243–252. doi:https://doi.org/10.1016/j.synbio.2017.11.002 (2017).
5. Moloney, M. G. Natural Products as a Source for Novel Antibiotics. *Trends in Pharmacological Sciences* **37,** 689–701. doi:https://doi.org/10.1016/j.tips.2016.05.001 (2016).
6. Jin, W., Barzilay, R. & Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. *arXiv* (2018).
7. Rarey, M. & Dixon, J. S. Feature trees: a new molecular similarity measure based on tree matching. *Journal of Computer-Aided Molecular Design* **12,** 471–490 (1998).
8. Vaswani, A. *et al. Attention is all you need* in *Advances in Neural Information Processing Systems* (2017), 5998–6008.
9. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* (2018).
10. Rives, A. *et al.* Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv,* 622803 (2019).
11. Rao, R. *et al.* Evaluating Protein Transfer Learning with TAPE. *bioRxiv.* doi:10.1101/676825 (2019).
12. Luo, Y. *et al.* Evolutionary context-integrated deep sequence modeling for protein engineering. *bioRxiv* (2020).
13. Riesselman, A. J. *et al.* Accelerating Protein Design Using Autoregressive Generative Models. *bioRxiv,* 757252 (2019).
14. You, Y. *et al. Large batch optimization for deep learning: Training bert in 76 minutes* in *International Conference on Learning Representations* (2019).
15. Nguyen, T. Q. & Salazar, J. Transformers without tears: Improving the normalization of self-attention. *arXiv* (2019).

16. Huang, P.-S., Boyken, S. E. & Baker, D. The coming of age of de novo protein design. *Nature* **537,** 320–327. doi:10.1038/nature1994 (2016).

17. Dou, J. *et al.* Sampling and energy evaluation challenges in ligand binding protein design. *Protein Science* **26,** 2426–2437. doi:10.1002/pro.3317 (2017).

18. Garcia-Borrás, M., Houk, K. N. & Jiménez-Osés, G. Computational design of protein function. *Computational Tools for Chemical Biology* **3,** 87. doi:10.1039/9781788010139-00087 (2017).

19. Romero, P. A. & Arnold, F. H. Exploring protein fitness landscapes by directed evolution. *Nature Reviews Molecular Cell Biology* **10,** 866–876. doi:10.1038/nrm2805 (2009).

20. Jeske, L., Placzek, S., Schomburg, I., Chang, A. & Schomburg, D. BRENDA in 2019: a European ELIXIR core data resource. *Nucleic Acids Research* **47,** D542–d549 (2018).

21. Liu, Y. *et al.* Roberta: A robustly optimized bert pretraining approach. *arXiv* (2019).

22. Cornish-Bowden, A. Current IUBMB recommendations on enzyme nomenclature and kinetics. *Perspectives in Science* **1,** 74–87 (2014).

23. Hadadi, N., MohammadiPeyhani, H., Miskovic, L., Seijo, M. & Hatzimanikatis, V. Enzyme annotation for orphan and novel reactions using knowledge of substrate reactive sites. *Proceedings of the National Academy of Sciences USA* **116,** 7298–7307 (2019).

24. Li, Y. *et al.* DEEPre: sequence-based enzyme EC number prediction by deep learning. *Bioinformatics* **34,** 760–769 (2017).

25. Dalkiran, A. *et al.* ECPred: a tool for the prediction of the enzymatic functions of protein sequences based on the EC nomenclature. *BMC Bioinformatics* **19,** 334 (2018).

26. Ryu, J. Y., Kim, H. U. & Lee, S. Y. Deep learning enables high-quality and high-throughput prediction of enzyme commission numbers. *Proceedings of the National Academy of Sciences USA,* 201821905 (2019).

27. Strodthoff, N., Wagner, P., Wenzel, M. & Samek, W. UDSMProt: Universal Deep Sequence Models for Protein Classification. *bioRxiv,* 704874 (2019).

28. Yang, K. K., Wu, Z. & Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nature Methods* **16,** 687–694. doi:10.1038/s41592-019-0496-6 (2019).

29. Bileschi, M. L. *et al.* Using Deep Learning to Annotate the Protein Universe. *bioRxiv.* doi:10.1101/626507 (2019).

30. Chen, H., Engkvist, O., Wang, Y., Olivecrona, M. & Blaschke, T. The rise of deep learning in drug discovery. *Drug Discovery Today* **23,** 1241–1250 (2018).

31. Butler, K. T., Davies, D. W., Cartwright, H., Isayev, O. & Walsh, A. Machine learning for molecular and materials science. *Nature* **559,** 547 (2018).

32. Kitchin, J. R. Machine learning in catalysis. *Nature Catalysis* **1,** 230 (2018).

33. Sanchez-Lengeling, B. & Aspuru-Guzik, A. Inverse molecular design using machine learning: Generative models for matter engineering. *Science* **361,** 360–365 (2018).

34. Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences* **28,** 31–36 (1988).

35. Jin, W., Yang, K., Barzilay, R. & Jaakkola, T. Learning multimodal graph-to-graph translation for molecular optimization. *arXiv* (2018).

36. Nguyen, T., Le, H. & Venkatesh, S. GraphDTA: prediction of drug–target binding affinity using graph convolutional networks. *bioRxiv,* 684662. doi:10.1101/684662 (2019).

37. Landrum, G. *RDKit: Open-source cheminformatics.*

38. Öztürk, H., Özgür, A. & Ozkirimli, E. DeepDTA: deep drug–target binding affinity prediction. *Bioinformatics* **34,** i821–i829 (2018).

39. Tian, S., Djoumbou-Feunang, Y., Greiner, R. & Wishart, D. S. CypReact: a software tool for in silico reactant prediction for human cytochrome P450 enzymes. *Journal of Chemical Information and Modeling* **58,** 1282–1291 (2018).

40. Sawayama, A. M. *et al.* A panel of cytochrome P450 BM3 variants to produce drug metabolites and diversify lead compounds. *Chemistry–A European Journal* **15,** 11723–11729 (2009).

41. Zanger, U. M. & Schwab, M. Cytochrome P450 enzymes in drug metabolism: regulation of gene expression, enzyme activities, and impact of genetic variation. *Pharmacology and Therapeutics* **138,** 103–141 (2013).

42. Brandenberg, O. F., Fasan, R. & Arnold, F. H. Exploiting and engineering hemoproteins for abiological carbene and nitrene transfer reactions. *Current Opinion in Biotechnology* **47,** 102–111 (2017).

43. Chen, K. & Arnold, F. H. Engineering new catalytic activities in enzymes. *Nature Catalysis,* 1–11 (2020).

44. Rydberg, P., Gloriam, D. E. & Olsen, L. The SMARTCyp cytochrome P450 metabolism prediction server. *Bioinformatics* **26,** 2988–2989 (2010).

45. Simulations Plus, I. *ADMET Predictor* Apr. 3, 2018.

46. Graves, A. Sequence transduction with recurrent neural networks. *arXiv* (2012).

47. Suzek, B. E. *et al.* UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics* **31,** 926–932 (2015).

48. Sterling, T. & Irwin, J. J. ZINC 15–ligand discovery for everyone. *Journal of Chemical Information and Modeling* **55,** 2324–2337 (2015).