Strategies and Tools for Machine Learning-Assisted Protein Engineering

Thesis by

Bruce James Wittmann

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY

Pasadena, California

2022

(Defended May 26, 2022)

Bruce James Wittmann
ORCID: 0000-0001-8144-9157

ACKNOWLEDGEMENTS

This thesis was only possible because of the immense support I have received both throughout my scientific career and otherwise. To start, I would like to thank my advisor, Prof. Frances Arnold, for everything that she provided during my thesis work. Thank you for your encouragement, input, and support throughout all my projects; the opportunities to meet, work with, and learn from exceptional scientists; and for overall providing an environment where I could learn and refine skills to confidently identify, pursue, and answer important research questions. My time in the Arnold Lab has been formative for me as a scientist.

I have had the opportunity to work with many fantastic people during my time at Caltech. I would like to thank in particular Dr. Sabine Brinkmann-Chen for always being available for a discussion, providing feedback on just about every piece of writing I produced at Caltech (including this thesis), and being a constant source of support throughout the often-tumultuous experience that is graduate school. I would also like to thank my direct mentors from the early stages of graduate school—Dr. Anders Knight, Dr. Jennifer Kan, and Dr. Zach Wu—who helped to set the stage for everything that was to come. Thank you all for welcoming me to the group, teaching me essential skills, and engaging in a number of direction-setting discussions. It was a pleasure collaborating with you on our various projects.

Speaking of collaborators, I would also like to thank all members of the Arnold Lab machine learning subgroup—past or present—that I have had the fortune to work with. In particular, I would like to extend my sincerest thanks to Kadina Johnston and Patrick Almhjell. You two are not only amazing coworkers, but amazing friends as well. Kadina, it was fantastic to build the machine learning subgroup up with you. It is quite astounding how far it has come when I think back to where it all started, and I am excited to see where it goes next. Patrick, I always appreciated our many discussions regarding data, programming, science in general, and more. You are one of the first people I met at Caltech, and it has been a pleasure to work alongside and learn from you these last five years. Both of you, thank you for helping to build evSeq into what it is and, more so, just making it an overall fun project to work on.

There are a number of people outside of the Arnold Lab—both at Caltech and elsewhere—who have been critically influential to my graduate school research experience as well. Thank you, Prof. Yisong Yue for your extremely helpful advice during the development of ftMLDE and a number of other machine learning projects that have since been created in collaboration. Thank you, Dr. Eric Horvitz not only for the opportunity to intern at Microsoft, but also for your encouragement, support, and input while I undertook what became one of the major projects described in this thesis. Similarly, I must thank everyone else that I worked with during my time at Microsoft—Dr. Kevin Yang, Umesh Madan, Dr. Ava Soleimany, Paul Koch, and Sanaa Mansoor—for their support throughout the internship.

Scientific collaborations aside, none of this would have been possible without the amazing friends I have made along the way. Dr. Austin Dulaney, you could be in both this paragraph and the previous one. On the practical side, thank you for your help in making the significant computational upgrades that enabled much of the work described in this thesis; thank you also for being a sounding board to many of my new machine learning ideas. Outside of work, our early morning workouts, late night hangouts, and any-time-of-day coffee breaks are defining of my graduate school experience, and were among the things I enjoyed most during my time at Caltech. Dr. Nicholas Porter, likewise, you could also show up in either this or the previous paragraph. Even though the projects we worked on together were far too much for two people to handle, I had a blast (and I think everyone else in the lab knew we had a blast too). Thank you for keeping things light and fun through some of the most stressful times. Patrick Almhjell and Kadina Johnston, again, thank you for your support both in and outside of the lab. Dr. Ella Watkins-Dulaney, Dr. Anders Knight, and Dr. Silken Jones, thank you—alongside Austin, Nick, Patrick, and Kadina—for all the good times with our various get-togethers (virtual or otherwise). To all of you, graduate school would not have been what it was without you—I am lucky to have met you all and I am grateful to have you as my friends.

To my family, thank you for always being a constant source of love and support. To my brothers, Tom and Freddie Wittmann, the two of you keep me going more than you probably realize. Thank you for always being a grounding presence and for helping me keep a more balanced perspective on things. To my parents, Kate and Peter Wittmann, a few sentences could never be enough to thank you appropriately. Thank you for always being available for a call or a visit; for taking interest in and sharing my papers, projects, and talks; for providing advice on topics ranging from day-to-day challenges to big-picture career decisions; and for overall being loving and supportive. I am where I am because of all you have done for me, and I am who I am because of all you have taught me. Thank you both for everything.

Finally, I would like to thank my partner, Ali Goodyear. Ali, a few sentences could never be enough to thank you properly either. All the way back in 2014, when we decided that we would go to graduate school, I do not think that either of us could have imagined everything that was to come. Throughout all of it—from the highest highs to the lowest lows and everything in between—thank you for being by my side. Thank you for celebrating with me when times were good and for keeping on believing in me (and helping me believe in myself) when times were bad. More so, though, thank you for being you when times were just times—for the simple, day-to-day moments that made this experience together one I will always cherish. It has been quite the journey, and I could not have done it without you.

ABSTRACT

Proteins perform critical roles in a growing list of human-devised applications, and as demands for new applications arise, new proteins must be engineered to meet them. Machine learning-assisted protein engineering (MLPE) has recently arisen as a new philosophy of protein engineering, promising to overcome many of the limitations of existing engineering strategies. Despite its promise, however, as a relatively new approach to protein engineering, MLPE faces many challenges that hinder its routine application. This thesis is focused on addressing a number of them. Chapter 1 provides a theoretical overview of protein engineering, introduces the core steps of a typical MLPE pipeline, and discusses the challenges that currently hinder MLPE's advancement. This chapter is written to be accessible to all members of the highly multidisciplinary audience that either use or develop MLPE tools, in turn providing a resource that eliminates the steep barrier to entry that can hinder broader participation in the field. Chapter 2 provides a solution to the challenge of applying MLPE to proteins whose fitness landscapes are dominated by "holes" (protein variants with zero or extremely low fitness). Using my development of the strategy "focused training machine learning-assisted directed evolution (ftMLDE)" as an example, I demonstrate how auxiliary information from protein sequence and structure can be used to navigate landscapes despite holes, in turn dramatically improving the efficiency of MLPE. Chapter 3 explores strategies for reducing the amount of sequence-fitness data needed for building MLPE models. Specifically, I detail the motivation behind and development of a new model designed to augment limited protein sequence-fitness datasets with information extracted from raw protein sequence and structure data. Finally, chapter 4 introduces "every variant sequencing" (evSeq), a collection of tools and protocols that enables extremely low-cost, routine collection of large protein sequence-fitness datasets. Not only does this technology drastically improve the financial feasibility of numerous MLPE applications, but it also potentiates the construction of a massive database of diverse protein sequence-fitness data, the likes of which would revolutionize our ability to engineer proteins with data-driven methods. Overall, the work described in this thesis advances both our understanding of MLPE and our ability to engineer proteins using it.

PUBLISHED CONTENT AND CONTRIBUTIONS

1. **Wittmann, B. J.**; Johnston, K. E.; Wu, Z.; and Arnold, F. H. (2021) Advances in Machine Learning for Directed Evolution. *Curr. Opin. Struct. Biol. 69*, 11–18. https://doi.org/10.1016/j.sbi.2021.01.008.

B.J.W. conceptualized the structure of this review article and performed the necessary literature review. The article was written with input from all authors.

2. **Wittmann, B. J.**; Yue, Y.; and Arnold, F. H. (2021) Informed Training Set Design Enables Efficient Machine Learning-Assisted Directed Protein Evolution. *Cell Syst. 12*, 1026-1045.e7. https://doi.org/10.1016/j.cels.2021.07.008.

B.J.W. conceptualized the research study, devised methodology for performing the research, wrote software to enable the research and ensure its reproducibility, validated the software, performed all experiments reported in the study, analyzed data from those experiments, validated the results from the experiments, and wrote the manuscript with feedback from other authors.

3. **Wittmann, B. J.**; Johnston, K. E.; Almhjell, P. J.; and Arnold, F. H. (2022) evSeq: Cost-Effective Amplicon Sequencing of Every Variant in a Protein Library. *ACS Synth. Biol. 11*, 1313–1324. https://doi.org/10.1021/acssynbio.1c00592.

B.J.W. conceptualized the method reported in this manuscript, wrote software and developed laboratory protocols to enable use of the method, validated the supporting software, and built the combinatorial DNA libraries reported in the work. The article was written with input from all authors.

PUBLISHED CONTENT NOT INCLUDED IN THESIS

† denotes equal contribution

1. Wu, Z.; Kan, S. B. J.; Lewis, R. D.; **Wittmann, B. J.**; and Arnold, F. H. (2019) Machine Learning-Assisted Directed Protein Evolution with Combinatorial Libraries. *Proc. Natl. Acad. Sci. 116*, 8852–8858. https://doi.org/10.1073/pnas.1901979116.

B.J.W. built DNA libraries with Z.W. as well as analyzed data with Z.W., S.B.J.K., and R.D.L. The article was written primarily by Z.W. with feedback provided by all other authors including B.J.W.

2. **Wittmann, B. J.**;† Knight, A. M.;† Hofstra, J. L.; Reisman, S. E.; Kan, S. B. J.; and Arnold, F. H. (2020) Diversity-Oriented Enzymatic Synthesis of Cyclopropane Building Blocks. *ACS Catal. 10,* 7112–7116. https://doi.org/10.1021/acscatal.0c01888.

B.J.W. performed evolution experiments for the engineering of the *trans*-cyclopropylboronate lineage, performed the final validation experiments for both lineages, and isolated the cross-coupling products and confirmed their identities by NMR and mass spectrometry. B.J.W. and A.M.K. contributed equally to this work, and both authors took primary responsibility for writing the manuscript with feedback provided by other authors.

3. Dallago, C.; Mou, J.; Johnston, K. E.; **Wittmann, B. J.**; Bhattacharya, N.; Goldman, S.; Madani, A.; and Yang, K. K. (2021) FLIP: Benchmark Tasks in Fitness Landscape Inference for Proteins. *bioRxiv*. https://doi.org/10.1101/2021.11.09.467890.

B.J.W. assisted with conceptualization of the work, selection of the appropriate datasets to use as benchmarks, calculation of baseline results, and writing the manuscript.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Active Learning: A machine learning strategy where a model is iteratively updated as new data become available. A common strategy is to (1) use a model to propose a set of datapoints to collect, (2) collect that data, (3) update the model with that data, and then (4) iterate through steps 1 – 3 until a desired engineering goal is achieved.

Amplicon: A fragment of DNA resulting from a PCR.

Benchmark Task: A standardized task that is used to compare the effectiveness of different machine learning models or approaches. Typically, a benchmark task consists of predefined sets of training, validation, and testing data. The most valuable benchmark tasks will test the practical utility of newly proposed models and approaches for solving real-world problems.

Contact Disruption: The number of contacts in a protein's 3D structure that are broken upon mutation, where a "contact" is simply an interaction between two amino acids.

Cross-Validation Error: The mean validation error of all folds tested in k-fold cross-validation.

Cross-Validation Indices: An "index" in programming is an integer that gives the position of an object in a list-like ordering of data. In this thesis, I use the term "cross-validation indices" to refer to a set of indices that give the positions of all datapoints used in each fold of k-fold cross-validation. Two experiments using the same cross-validation indices thus split the data in the same way for performing k-fold cross-validation.

Degenerate Oligonucleotide: A pool of short fragments of DNA. Most often, the sequences of those short fragments will differ at only a single or a few positions. These are frequently used to build protein mutant libraries in a highly multiplexed fashion.

Directed Evolution: A protein engineering strategy that proceeds through rounds of mutagenesis and screening, fixing the most-improved protein mutant at each round to greedily arrive at an optimized variant.

Dataset Shift: A phenomenon where testing data comes from a different (or shifted) distribution relative to the training data. This generally leads to deteriorating performance of machine learning models.

Dry Lab: A research environment where experiments are computationally performed.

Encoding: Machine learning models operate on numerical inputs. As a result, non-numerical inputs (e.g., a protein sequence) must first be represented—or, "encoded"—using a set of numbers. In this thesis, I use the term "encoding" to refer to (1) the general set of features that are used for representing protein sequences, (2) a vector of features that describes a

specific protein sequence, or (3) the process of converting a non-numerical input to a numerical one.

Exploration-Exploitation Tradeoff: When searching for optimal solutions with limited resources, a decision must be made regarding how to best allocate those resources to balance taking advantage of existing knowledge (exploitation) and collecting new knowledge (exploration). This balance is known as the exploration-exploitation tradeoff. Exploitation is likely to yield many good, yet perhaps suboptimal solutions; exploration, by contrast, will yield more varied solutions, but perhaps some that are better than would have otherwise been found focusing on exploitation alone.

Feature: A numerical value that describes some aspect of an input to a machine learning model. A set of features makes up an encoding.

Functional Screen: Any process that evaluates a protein's ability to perform a task.

Generative Model: A machine learning model that learns an underlying probability distribution from data.

Global Optimum: A function can have multiple optima. The global optimum is the most extreme of all optima. On a protein fitness landscape, the global optimum would be defined by the sequence with the highest possible fitness.

Homologous Proteins: Two proteins are "homologous" if they share a common ancestor.

Kernel (in the context of Gaussian processes): The function used to calculate each entry of the covariance matrix of a Gaussian process.

K-Fold Cross-Validation: The performance of a machine learning model is typically described using both a training error and validation error. Training error represents the ability of a model to predict the correct values for the training data and is directly used to learn model parameters. Validation error represents the ability of the model to predict the correct values for data not used to train it. Validation error is used to select the optimal model architecture for a task to avoid selecting an architecture that has obtained good training error by overfitting to noise or other idiosyncrasies in the training data. When working with limited data, however, it is often undesirable to set aside a set of data for validation error calculation, as that data will by definition not be used for training. In such a case, k-fold cross-validation can be used. In this procedure, data is split into k chunks, or "folds." Then, a model is trained using k-minus-one folds and validation error is calculated on the held-out fold. A new model is then instantiated and trained on a different combination of k-minus-one folds, again calculating a validation error on the held-out fold. The procedure iterates until all folds have been used for calculating validation error and a mean "cross-validation error" is returned. Model architectures that achieve good cross-validation error are assumed to be the most effective for a given task.

Labeled Data: Data that consists of both x- and y-values. The y-value is typically referred to as the "label" of the x value. For proteins, for example, labeled data might consist of a set of protein sequences with associated fitness scores. The fitness scores would be the labels of the sequences.

Learned Embedding: For the purposes of this thesis, a "learned embedding" is an automatically learned encoding derived from unlabeled data.

Learning Objective: Machine learning models are trained to optimize some loss function (e.g., they might be trained to minimize mean-squared error). The loss function chosen for optimization is often referred to as the "learning objective" for a given training procedure.

Local Optimum: A function can have multiple optima. Each of these optima is considered a "local optimum" as they are optimal relative to their local environment. On a protein fitness landscape, for instance, a local optimum is defined by a sequence from which a single-step greedy walker would not be able to leave.

Machine Learning: A strategy for automatically building computational models from data.

Machine Learning Model: A mathematical function defined by some set of learnable parameters.

Machine Learning-Assisted Protein Engineering: An approach to protein engineering where machine learning strategies are employed to assist with the design or identification of new, useful proteins.

Masked-Token Prediction: A semi-supervised strategy derived from natural language processing. In this approach, a subset of items in a sequence of items are obscured ("masked"); the identities of these obscured items are then predicted using the context provided by the unobscured items.

Model Architecture: At a high level, the goal of machine learning is to fit a model (a function) to a dataset for the purpose of either (1) extracting useful information from that data or (2) making predictions on as-yet unseen data. The parameterization of the model (i.e., the structure of the formula that defines the model) is known as the "architecture" of that model.

Model Ensemble: A set of models, often with different model architectures.

Molecular Barcode: A unique DNA sequence element that encodes the identity of a sample in a population of DNA sequences.

Multiplex: To multiplex is to perform multiple operations in parallel. In this thesis, "multiplex" is primarily used in the context of molecular biology, where it typically refers to performing multiple molecular biology reactions in parallel.

Next-Token Prediction: A semi-supervised strategy derived from natural language processing. In this approach, the next item in a sequence of items is predicted using only information contained in preceding items.

Oligonucleotide: A short fragment of DNA.

One-Hot: A simple strategy for encoding categorical data. Each category is assigned an index in a vector. At this index, the vector has value "1"; at all other positions, it has value "0." As an example, for a protein of length L, a one-hot encoding would result in a matrix with shape L × 20. A given row would consist of 19 values of 0 and a single value of 1; the column containing that value of "1" would depend on the identity of the amino acid at the position in the protein represented by the row.

Parameter: A value that defines a mathematical model or function. In machine learning, the optimal numerical value for each parameter defining a model is learned from data during training.

Parent Protein: The protein used to initiate a protein engineering study. This is often incorrectly conflated with the "wild-type" protein, which has a different definition (see below).

Physicochemical: A concatenation of "physical" and "chemical." This term is often used as an adjective used to describe the physical and chemical nature of something.

Protein Activity: Depending on the context, this can be a synonym for either protein fitness or protein function, though it is most often used to describe fitness. When used as a quantity, it describes protein fitness (e.g., "the protein's activity was measured and found to be high"); when used as a quality, it describes protein function (e.g., "this protein's activity is to catalyze that reaction").

Protein Engineering: A field of study dedicated to the development of new proteins with useful functions.

Protein Fitness: How well a protein performs a specific function.

Protein Fitness Landscape: A conceptualization of the relationship between protein sequence and protein fitness. This is a surface in a high-dimensional space defined by the function f(sequence) = fitness.

Protein Function: The specific task that a protein performs.

Protein Library: A set of proteins. Typically, this set consists of protein mutants all derived from the same parent protein.

Protein Sequence: The ordering of amino acids that defines a protein.

Protein Variant: A synonym for a protein mutant. That is, a protein with at least one amino acid change relative to some parent protein.

Random Seed: Computers rely on pseudorandom number generators to approximate processes of randomness. A random seed can be fed into a random number generator to produce reproducible patterns of randomness. In other words, two random processes run with the same random seed will yield identical results.

Rational Design: An approach to protein engineering that aims to build computational models of protein fitness and function from physical and chemical principles. These models are then used to identify new and useful proteins.

Representation Learning: A machine learning strategy where useful features (representations) are learned from (typically) unlabeled data. This strategy is closely tied to self-supervised learning (a strategy often used to drive representation learning) and semi-supervised learning (where representation learning will typically make up the unsupervised stage of the workflow).

Self-Supervised Learning: A machine learning strategy where labels are automatically constructed from unlabeled data and then used to train a model.

Semi-Supervised Learning: Any machine learning strategy that involves both an unsupervised and a supervised learning phase. Most often, this refers to an approach where a representation learned from unlabeled data is used in a downstream supervised learning problem.

Sequencing Coverage: The number of returned reads from a next-generation sequencing experiment that map to a specific nucleotide on a reference sequence.

Starting Point Problem: Most protein engineering strategies assume that a protein with even the smallest amount of the desired activity is already known. The challenge of finding this protein is referred to as the "starting point problem."

Supervised Learning: Any machine learning strategy that learns from labeled data.

Training Data: The data used to train a machine learning model.

Training Error: A value that represents the ability of a model to predict the correct values for the data used to train it. Training error is optimized to learn model parameters during model training.

Training Set: The data used to train a machine learning model.

Transfer Learning: A machine learning strategy where information learned in one problem is used to assist in solving another. In other words, a machine learning strategy where information from one problem is transferred to another.

Unlabeled Data: Data that consists of x-values alone. For proteins, a dataset consisting of protein sequence data alone would be considered unlabeled.

Unsupervised Learning: Any machine learning strategy that learns from unlabeled data.

Validation Data: Data that is held aside during training of a machine learning model. After training, the validation set is used to calculate a validation error and evaluate how effectively the model learned to generalize beyond the training data.

Validation Error: A value that represents the ability of a model to predict the correct values for the data not used to train it.

Validation Set: Data that is held aside during training of a machine learning model. After training, the validation set is used to calculate a validation error and evaluate how effectively the model learned to generalize beyond the training data.

Wet Lab: A research environment where experiments are performed by manipulating physical entities. For instance, a protein engineering wet lab will involve working with cultures of organisms, running chemical reactions, etc.

Wild-Type Protein: A protein whose sequence is found in nature. If multiple mutants of a protein sequence are found in nature, then the wild type is the typical (dominant) one. A wild-type protein can be a parent protein, but not all parent proteins are wild-type proteins.

Zero-Shot Prediction: For the purposes of this thesis, “zero-shot prediction” refers to a prediction made using a model that can be trained or used without the need for additional experimental collection of data (i.e., collecting additional labeled data).

ABBREVIATONS

AAV	Adeno-Associated Virus
BERT	Bidirectional Encoder Representations from Transformers
BFD	Big Fat Database
bp	Base Pair
CASP	Critical Assessment of protein Structure Prediction
CNN	Convolutional Neural Network
DE	Directed Evolution
DMS	Deep Mutational Scanning
DNA	Deoxyribonucleic Acid
EDTA	Ethylenediaminetetraacetic acid
ESM	Evolutionary Scale Modeling
evSeq	Every Variant Sequencing
FLIP	Fitness Landscape Inference for Proteins
ftMLDE	Focused Training Machine Learning-Assisted Directed Evolution
GB1	G Protein Domain B1
GFP	Green Fluorescence Protein
GP	Gaussian Process
GUI	Graphical User Interface
h	Hour
hrs	Hours
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IPTG	Isopropyl-β-D-Thiogalactoside
kb	Kilo-bases
KL Divergence	Kullback–Leibler Divergence
LB	Luria Broth
LSTM	Long Short-Term Memory
mg	Milligram
ML	Machine Learning
mL	Milliliter
MLDE	Machine Learning-Assisted Directed Evolution
MLPE	Machine Learning-Assisted Protein Engineering
mM	Millimolar
MSA	Multiple Sequence Alignment
MSE	Mean Squared Error
NDCG	Normalized Discounted Cumulative Gain
NGS	Next Generation Sequencing
NLP	Natural Language Processing

nmole	Nanomole
PCR	Polymerase Chain Reaction
PDB	Protein Data Bank
PLP	Pyridoxal 5'-Phosphate
ResNet	Residual Network
*Rma*NOD	*Rhodothermus marinus* Nitric Oxide Dioxygenase
RMSD	Root-Mean-Squared Deviation
rpm	Revolutions Per Minute
rxn	Reaction
Spearman ρ	Spearman Rank Correlation Coefficient
TAPE	Tasks Assessing Protein Embeddings
TB	Terrific Broth
*Tm*TrpB	*Thermotoga maritima* Tryptophan Synthase β-Subunit
TrpB	Tryptophan Synthase β-Subunit
UCB	Upper Confidence Bound
VAE	Variational Autoencoder
XGBoost	Extreme Gradient Boosting
μg	Microgram
μL	Microliter
μM	Micromolar

Chapter 1

MACHINE LEARNING-ASSISTED PROTEIN ENGINEERING

Material from this chapter appears in **Wittmann, B. J.**; Johnston, K. E.; Wu, Z.; and Arnold, F. H. (2021) Advances in Machine Learning for Directed Evolution. *Curr. Opin. Struct. Biol.* *69*, 11–18. https://doi.org/10.1016/j.sbi.2021.01.008.

Abstract

Machine learning can accelerate protein engineering by allowing researchers to move expensive laboratory screens *in silico*. Driven by increased computational power, the continued decline in the cost of DNA sequencing, and rapid improvements in laboratory screening technologies, machine learning-assisted protein engineering (MLPE) is experiencing a rapid growth in popularity and applicability. As a highly multidisciplinary approach, interest in MLPE continues to grow from both the broader machine learning and broader protein engineering communities; unfortunately, however, because there is rarely cross-disciplinary training between these two communities, there can be a steep barrier to entry for anyone interested in using, developing, or understanding MLPE methods. The goal of this chapter is to eliminate that barrier, introducing the core concepts of MLPE in a way that should be accessible to all readers, regardless of academic discipline. I begin with a broad overview of protein engineering before moving into a discussion of a typical MLPE pipeline; I then end the chapter by highlighting a number of outstanding MLPE challenges. Throughout this chapter, I discus concepts from a primarily higher-level intuitive view, leaving presentation of most technical details to provided references and instead focusing on providing the conceptual framework needed to understand both later chapters of this thesis and the field of MLPE as a whole. This chapter should serve as a central resource for anyone interested in MLPE.

1.1 Proteins and Protein Engineering Philosophies: A Brief Overview

Life, in all its complexity, is built from the simplest building blocks. DNA encodes the information needed to construct all living things, yet it is composed of combinations of just four unique nucleotides. Proteins, which result from the instructions of DNA, perform central functions in almost all processes of life, yet are almost universally built from just 20 distinct amino acids.

Complexity arises from simple building blocks via the staggering number of combinations that can be made from them. For instance, to build a protein, amino acids are connected one after the other in a long chain. Considering an average protein with a chain length of 300, with 20 amino acid options at each position in the chain there are $20^{300} \approx 10^{390}$ different possible configurations of amino acids, which is approximately 10^{310} times more possibilities than the estimated number of atoms in the observable universe. What (if anything) a protein does and how well it does it is, to a first approximation, determined by its configuration ("sequence") of amino acids; given the beyond astronomical number of potential sequences, the enormous range of functions performed by proteins in life is perhaps unsurprising.

The diversity of proteins seen in life today is the result of billions of years of evolution, during which time only a negligible fraction of the space of all possible protein sequences has been explored (given the size of protein sequence space, full exploration will never be accomplished). This suggests a large number of potentially useful protein functions left to be discovered, certainly by life, but also by human engineers. Indeed, many of the characteristics that make proteins vital to life's processes make them valuable for human-devised applications as well. Due to their selectivity, efficiency, and numerous other advantageous traits, proteins now play integral roles in industries ranging from pharmaceuticals to consumer products, materials, food, and fuels, and their importance is expected to continue to grow.[1–4]

Just as life requires effective strategies for producing new proteins (e.g., to respond to changing environmental stressors), so too do humans, as new proteins must often be developed in response to new or changing demands. Unlike life, however, which sees the

arrival of new proteins as a result of evolution over long periods of time, we need strategies—protein engineering strategies—that can identify proteins with a desired function rapidly and efficiently.

Protein engineering strategies have typically been broadly divided into two complementary yet different philosophies: rational design and directed evolution.[5–11] Rational design posits that chemical and physical laws can be used to intelligently engineer proteins for a target function.[5–8] When amino acids are chained together in a protein sequence, they will interact with each other and the surrounding environment to fold the protein sequence into a 3-dimensional structure.[12] Because the twenty amino acids have distinct physical and chemical properties, the precise positioning of the different amino acids within that structure dictates the nature and type of interactions the protein can make with other molecules, in turn determining its function. Rational design asserts that if we could understand (1) what 3D configuration of amino acids is needed to perform a specific function and (2) what amino acid sequence will give us that configuration, then we can rapidly identify proteins useful for a new task.

In principle, rational design is the perfect protein engineering strategy, allowing us to create new proteins for new tasks at will. In practice, however, and despite a number of success stories, it tends to be challenging to implement. For one, it is not always clear what 3D configuration of amino acids will yield a desired function; at least some knowledge of the chemical mechanism underlying the function is required to propose one, and this information is often unavailable, particularly when developing proteins with completely new functionality. Additionally, the relationship between sequence and structure is complex; it is difficult to precisely predict the structure into which a given sequence will fold, so even if an ideal 3D amino acid configuration for a given function is known, identifying a sequence that will yield it (if one exists at all) is challenging. As a result of these limitations, particularly for protein functions resulting from complex or unknown mechanisms, rational design is typically not considered a general-purpose engineering strategy.

In contrast to rational design, directed evolution requires no understanding of a system's structure-function relationship to be effective.[9–11] It accomplishes this by borrowing the protein engineering algorithm of nature: evolution. In life, new protein functions emerge as a result of changing environmental pressures. While proteins are generally optimized for a specific function, they also often exhibit slight ability to perform others, a trait that is commonly called "promiscuity."[13] If, by chance, an existing protein has even the slightest level of promiscuous activity for a function that provides even the smallest competitive advantage to its host organism in the face of a new environmental pressure—for instance, by providing a degree of resistance to some newly introduced toxin—then that organism is more likely to survive and propagate than others. In each new generation of organisms, random mutations can occur that change the amino acid sequence of that protein, and in turn its ability to perform this advantage-granting function. Some changes will diminish or eliminate protein function while others will improve it. Those organisms hosting a protein with improved function will have a competitive advantage and thus be more likely to propagate than others; this "natural selection" will lead to an enrichment of proteins with improved function in the next generation. Over many generations of mutation followed by natural selection, a new protein will gradually emerge that is optimized for the once-promiscuous function. Notably, there was no rational intervention during this evolutionary process. There was no need to understand the mechanistic basis for the observed function. Random mutations to an existing protein sequence in the presence of an environmental pressure were all that was needed.

Like in natural evolution, a directed evolution workflow proceeds by iterating through cycles of mutagenesis and selection for improved function (Figure 1-1A).[9–11] The workflow begins by identifying a protein with even the slightest ability to perform a desired function. Next, mutants (also commonly referred to as "variants") of this protein are produced and evaluated; the mutant with the greatest activity (greatest ability to perform the function) is identified and used as the starting point for the next generation, and the process iterates until a protein with the desired level of activity is identified. For the same reasons as in natural evolution, this process does not require detailed mechanistic understanding of the system being

engineered to be effective. Unlike in natural evolution, however, because humans decide which protein is used as the starting point (the "parent") for each new generation, the protein function being optimized does not need to be tied in any way to survival of an organism; so long as a quantifiable laboratory assay (a "screen") of protein function can be developed, that function can be optimized by directed evolution. Not only, then, is directed evolution simple to employ, it also highly generalizable. For both of these reasons, directed evolution is typically regarded as the most successful general-purpose protein engineering strategy to date, with its wide-reaching contributions to academia, industry, and society as a whole recognized by the award of (one-half of) the 2018 Nobel Prize in Chemistry.

Despite its undeniable success and effectiveness, directed evolution has limitations. For instance, it cannot be performed unless a protein with at least some ability to perform the desired function is already known—it can only optimize existing, known function. This "starting point problem" can present a significant bottleneck to the development of new protein functions, as existing strategies for solving it tend to be applicable only in limited situations or else are generally unreliable.[14] In addition to the starting point problem, directed evolution's reliance on a functional screen necessitates extensive laboratory characterization of produced protein variants, making the process time- and resource-intensive and presenting a bottleneck for engineering many protein functions where screening more than a few hundred or thousand variants would be expensive.

In an effort to overcome some of the limitations of directed evolution and rational design, recent years have seen the arrival of a third philosophy of protein engineering: machine learning-assisted protein engineering (MLPE).[15–24] From an applications point of view, machine learning (ML) can broadly be defined as "a strategy for automatically building computational models from data," where a "model" can be thought of as some function that maps an input representation to an output representation.[21,25,26] This model will be defined by some set of parameters, and the goal of ML is to identify ("learn") the ideal set of parameters for performing that mapping. For instance, a linear transformation ($\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$) can act as a simple model, where an input vector $\mathbf{x}$ is mapped to an output vector $\mathbf{y}$ by

parameters contained in matrix **W** and vector **b**. If this linear model were used in a machine learning application, the goal would be to use known pairs of **x** and **y** to learn the parameter values that are optimal (though perhaps not perfect) for transforming **x** to **y**. These learned parameters could then be used to predict the **y** associated with a previously unseen **x**. This simple ML example with a linear model will likely be familiar to most people—anyone who has ever added a line of best fit to a plot (e.g., using Microsoft Excel) of the form y = mx + b has performed it, only here the goal would have been to find scalar forms of **W** and **b** ("m" and "b") that best transform a scalar x into a scalar y.

In MLPE applications, the goal is to build a model that can map a protein sequence to its ability to perform some function. The model is first constructed using existing protein data, then used in place of a laboratory screen to evaluate previously unseen proteins, drastically reducing the experimental screening burden (Figure 1-1B). In a way, this approach is similar to rational design, as improved proteins are predicted rather than stochastically identified through laboratory screening as in directed evolution. Importantly, however, because the model is learned from patterns in the data, MLPE does not require any understanding of the system to be effective, thus avoiding rational design's key limitation. MLPE thus has the advantages of both rational design and directed evolution: it avoids extensive screening burdens (as in rational design) and can also be used to engineer arbitrary protein functions (as in directed evolution).

Despite its potential, as a relatively new approach to protein engineering, there remain numerous open questions and practical challenges to applying MLPE. For instance, the effectiveness of ML in general depends heavily on the quality and quantity of the data available for training models, and the more high-quality data we can collect, the better our ML models will perform. Just as laboratory screening capacities act as a bottleneck for directed evolution, they also limit our ability to gather data needed to train the models used in MLPE. In other words, we will always have a limited budget for gathering new data for MLPE. How, then, should we most effectively use this limited screening budget? How do we choose which new data to collect that will lead to the most effective MLPE models? How

do we most effectively use data that has already been collected? Alternatively, can we develop laboratory methods that increase our screening budget? The focus of this thesis is to answer these questions.

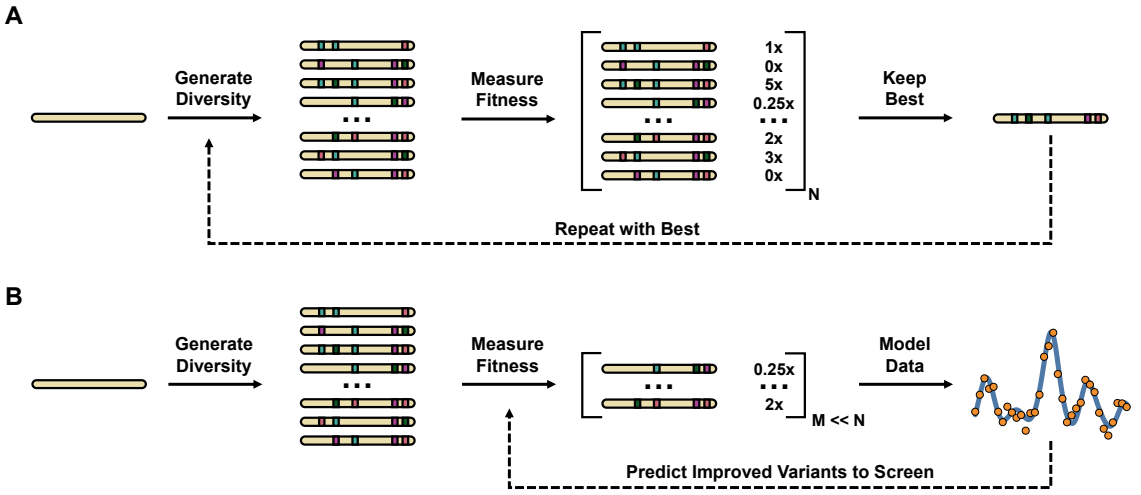


Figure 1-1. Example workflows of (A) traditional directed evolution and (B) supervised machine learning for directed evolution. Both workflows begin by identifying a protein with activity for a target function. Once the starting point is identified, diversity is introduced by mutagenesis and resulting variants are screened for function. (A) In traditional directed evolution, many variants are screened and the best variant is then fixed as the parent for the next round of mutagenesis/screening. (B) When applying supervised machine learning to directed evolution, fewer variants are screened. Using the resulting sequence-function data, a function is fit that relates protein sequence to protein fitness (e.g., for f(x) = y, "x" is the protein sequence and "y" is the protein fitness). This function can be used to predict the fitnesses of variants not experimentally evaluated or to propose a new set of variants to screen in the next round of evolution.

The remainder of this chapter is dedicated to presenting (1) an overview of the MLPE pipeline, highlighting recent advances at each of the key steps and (2) a brief discussion of the core challenges currently faced by the MLPE community. This chapter does not need to be read to understand later ones, but should provide greater context for the problems addressed in them. My goal is that anyone who reads this chapter—regardless of academic background—will gain a basic understanding of why ML is applied to protein engineering, how ML is applied to protein engineering, the core considerations that must be made at each step of an MLPE endeavor, and where MLPE still has room for improvement. It will not serve as a manual for applying ML to protein engineering, instead providing more of a high-

level overview and refraining, where possible, from deep dives into specifics. There are already a number of resources published for the interested reader that can serve as manuals for applying ML to protein engineering,[16,18,21] and I will also provide references throughout that give significantly more detail on any of the discussed topics.

1.2 Machine Learning-Assisted Protein Engineering in Practice

This section describes the typical steps of an MLPE pipeline, using recent advances to highlight the considerations that must be made at each of them. Because this thesis is primarily concerned with how we can most effectively use ML to optimize the activity of a given protein, that is the lens through which I will present it. However, it is important to note that the MLPE considerations introduced in the subsections below are generally relevant to all of its applications.

1.2.1 Training Data Selection for Mapping Protein Fitness Landscapes

The relationship between a protein's amino acid sequence and the extent to which it can perform a desired function (its "fitness") is often conceptualized as a surface in high-dimensional space called a "protein fitness landscape."[27–29] Within this conceptual framework, the goal of protein engineering is to search a fitness landscape to identify amino acid sequences optimized for performing specific functions—that is, the goal is to identify fitness peaks. Exactly how these peaks are found varies by protein engineering strategy. Directed evolution, for instance, can be thought of as a greedy uphill walk on a fitness landscape: starting from a parent protein low on a fitness peak, it iteratively identifies proteins with improved fitness until a fitness optimum is reached. Rational design and MLPE take a different approach, building the function *f(sequence)* = *fitness*,[a] then using that function as a map to computationally explore a fitness landscape and (in principle) identify the global optimum. As discussed in the first section, the key difference between rational

[a] Strictly speaking, rational design usually attempts to build a function of the form rank(*f(sequence)*) = rank(*fitness*), where a score *correlated* to fitness is predicted, not fitness itself. As will be discussed later, many MLPE strategies also attempt to build a function of this form.

design and MLPE is that while rational design builds the landscape map from chemical and physical principles, MLPE builds it from data.

The data used to build an ML model is called "training data." In most MLPE applications, training data consist of either sequence-annotation pairs, or simply sequences on their own. When data are unannotated (e.g., just sequences), then they are considered "unlabeled"; when data are annotated (e.g., sequence-annotation pairs), then they are considered "labeled." The exact annotation used in labeled data will vary by application. For instance, if I wanted to train an ML model to predict the host organism of proteins, then I would ideally train it using a labeled dataset of sequence-*organism* pairs. If I wanted to build a map of a protein fitness landscape for MLPE, then I would ideally gather a labeled training dataset of sequence-*fitness* pairs. Gathering a training dataset of sequence-fitness pairs generally involves (1) building a set (a "library") of protein sequences and then (2) experimentally evaluating their fitnesses. There are many different strategies for building protein libraries, the details of which I will leave to the referenced sources;[11,30] the experiment for evaluating protein fitness will, of course, depend on the definition of "fitness" for the protein system being studied.

In general, the data used to train an ML model determines what it learns and, by extension, in what situations it can be used to make effective predictions. ML models tend to be more effective at interpolation than extrapolation and will thus typically perform best when used to make predictions in the same domain as the data used to train them. For general ML applications, this typically translates to collecting maximally diverse training data that best cover the space being modeled. For MLPE, this means that training data with maximal protein sequence diversity will be most informative for modeling a fitness landscape:[b] the more diverse the training sequences are, the more of the fitness landscape that is covered by the training data and the less a model must extrapolate to previously unseen regions of sequence space. Randomly collecting sequences can thus be a valuable strategy for

[b] Strictly speaking, training data that maximizes diversity in the *encoded* sequences will be the most informative. More information is provided on protein encoding in Section 1.2.2.

constructing an MLPE training dataset, as this will on average result in the collection of highly diverse sequences.

While building a perfect map of a fitness landscape would be ideal for model-guided engineering, it is not feasible given our limited ability to collect experimental data. More complex fitness landscapes considering larger sections of sequence space require more data to model, and a small amount of randomly selected labeled training data may be spread too thinly across sequence space to build a comprehensive map.[31] For instance, as more mutations are made to a protein, the probability that it retains function decreases exponentially,[32] so fitness landscapes consisting of combinations of mutations at multiple positions ("combinatorial landscapes") tend to be dominated by protein variants with zero or extremely low fitness. These variants are commonly referred to as "holes" in the fitness landscape as they only provide information on which mutations destroy protein fitness; they do not provide information about the *extent* to which a mutation impacts protein fitness. By simple probability, a small, randomly drawn labeled training dataset from a hole-filled landscape will itself tend to be dominated by holes and contain limited if any information about the relatively rare peaks in the fitness landscape. A model trained on this data will thus also lack information about the fitness peaks, which is a problematic result from the point of view of protein engineering because the peaks are where we will find protein variants with improved fitness and are therefore what we are most interested in mapping.

In general, the goal of MLPE is to reduce laboratory screening burdens. It is thus desirable to develop data-efficient MLPE strategies that maximize our ability to identify improved protein variants while minimizing the need for experimental collection of additional training data. As part of my thesis work (as detailed in Chapter 2 and in the provided reference), I proposed and then demonstrated that, if training data are expensive to collect, it can be advantageous to build focused training datasets that are biased toward protein variants believed *a priori* to be higher in fitness rather than to build training datasets from randomly selected protein sequences.[33] Continuing the example from the previous paragraph, this would mean using some source of prior knowledge of the protein fitness landscape being

modeled to avoid including holes in the training data. The logic behind this approach is that it is more important to be able to identify the highest-fitness variants from the set of high-fitness variants than the lowest-fitness variants from the set of low-fitness variants; thus, we should aim to model (potentially) higher-fitness regions of the protein fitness landscape at higher resolution (which is enabled by biasing data collection to these regions) and lower-fitness regions of the protein fitness landscape at lower resolution (which results from collecting less data from these regions). This logic better aligns with the overall goals of MLPE, which is not to comprehensively map fitness landscapes, but to use ML to guide exploration of fitness landscapes to reach higher-fitness protein variants.

1.2.2 Protein Encoding Strategies and Semi-Supervised Learning

ML models work by performing mathematical operations on an input space to map it to an output space. For example, a simple linear model given by $\mathbf{y} = \mathbf{W}\mathbf{x}$ transforms an input vector $\mathbf{x}$ into an output vector $\mathbf{y}$ using the learned weights matrix $\mathbf{W}$. The ideal model for performing this mapping will vary by application, as it depends on the nature of the relationship between the two spaces. For the sake of keeping this chapter a big-picture introduction to MLPE, I will not go into the technical details of how the ideal model for a given problem is chosen, leaving that to referenced materials.[21,34] In general, however, simple relationships between input and output spaces can be described by simple models (those with fewer learnable parameters), while more complex relationships require more complex models. Importantly, as model complexity increases, the amount of training data needed to learn the relationship does as well, so ML is typically more efficient in applications where there is a simple relationship between input and output spaces.

In many ML applications, the input space is not—at least initially—clearly defined. For example, say the goal was to build an ML model that could predict the average rating given to movies by critics. The output space will clearly be defined by numerical labels like “score out of ten,” but movies are inherently non-mathematical objects, so what is the input space? To apply ML in cases like this, the input space must first be defined by describing each non-mathematical object with a set of numerical “features” (a process known as “encoding”). For

instance, a movie might be encoded using a set of features like "year of release," "film budget," etc. In general, it is ideal to choose features that are believed to be highly correlated with the target label, as this will set up a simpler relationship between input and output spaces and better enable data-efficient learning. We thus might expect "film budget" to be a particularly helpful feature for predicting critical ratings; it is unlikely to be as helpful, however, if our goal were to instead predict a less correlated label like "filming location."

Just like movies, protein sequences are also inherently non-mathematical objects, and must be encoded to be fed into ML models. A common encoding strategy applied to proteins is "one-hot encoding." In this strategy, each position in the protein is encoded by a vector; the vector will have 20 positions, all filled with "0" except for a single "1" whose position is determined by the identity of the amino acid in question. For instance, the amino acid "alanine" might be encoded by a vector with "1" in the first position, the amino acid "cysteine" might be encoded with "1" in the second position, and so on. While this encoding strategy is sufficient for training an ML algorithm, it is clearly uninformative. Amino acids exhibit a spectrum of different physical and chemical ("physicochemical") properties, with some more similar to one another than others. Yet, using a one-hot encoding, this information is not presented to the ML model. Instead, each amino acid is simply treated as a different category, all differing only in the placement of the "1" in their encoded representation.

Physics and chemistry ultimately determine what a protein does and how well it does it, so encoding strategies that better capture the physicochemical similarities and differences of amino acids can be expected to be more informative to an ML model. The most obvious way to capture such information in a protein encoding is to explicitly use numerical descriptors of the different amino acids' physicochemical properties. Projects such as the AAIndex have collected hundreds of such descriptors—some of which result from laboratory measurements of amino acid properties and others of which have been manually designed to describe amino acid qualities—that can be used in different combinations to create different physicochemically grounded protein encodings.[35,36] As shown in both Chapter 2 and other

works, the additional information provided by physicochemical features can indeed improve MLPE performance compared to using one-hot encoding.[19,33]

Something notable about both one-hot and physicochemical encoding strategies is that neither takes into account the greater context of the protein sequence: every amino acid is given the same encoding regardless of its position relative to other amino acids in the sequence. The function of each amino acid in a protein is, however, extremely context dependent. For instance, a serine in a protein's active site (the location where a chemical reaction occurs in a protein) will likely perform a very different role than a serine at that protein's surface. We might imagine, then, that an encoding scheme where the amino acid representation changes based on its context would be even more informative than a physicochemical one.

Unfortunately, it is not immediately clear how to explicitly capture context in the encodings of different amino acids. The number of possible contexts in which we can find a particular amino acid is only slightly smaller than the space of all possible proteins,[c] and so, unlike for context-independent one-hot or physicochemical encodings, we cannot simply write down a set of rules of the form "when amino acid X is in the presence of sequence Y, use encoding Z." Just as the relationship between amino acid sequence and fitness is complex, so too is the relationship between amino acid sequence and context-dependent encodings. Fortunately, both of these relationships can be mapped in the same way: by using machine learning.

In recent years, particular interest has arisen around training ML models to automatically build protein encodings using information extracted from large unlabeled protein datasets.[22,37,38] Unlike labeled protein data, unlabeled protein data are extremely inexpensive to produce. Indeed, drastic reductions in sequencing costs have led to a deluge of unlabeled sequence data, with hundreds of millions of naturally occurring protein sequences identified from various organisms now stored in online databases.[15,39–42] Importantly, all proteins found

[c] For a protein sequence of length N, there are 20^N possible protein sequences. An amino acid at a single position can be found in the context of all other possible amino acid combinations at all other positions. That is, it can be found in 20^{N-1} possible contexts.

in life today—and thus all proteins found in those databases of sequences—must follow some set of biophysical and evolutionary rules that allow them to be produced and carry out a biological function; otherwise, they would have been filtered out by evolution. By training models, which are often adapted from natural language processing (NLP),[43–47] on unlabeled protein sequences, the sequence constraints that result from these rules can be learned (Figure 1-2A, Figure 1-2B).[48–55] These models can then be repurposed to generate continuous vector representations of proteins known as "embeddings," which can be used for protein encoding (Figure 1-2C, Figure 1-2D). An effective protein embedding will capture information learned during pretraining and define the relationships between proteins within the context of learned sequence constraints. Whether or not an amino acid obeys a sequence constraint depends on its context, so it will have a different encoding depending on its relative location in a protein sequence.

The full process of (1) training an ML model on unlabeled data, (2) using that model to encode proteins, and (3) training another model to map between those encoded proteins and a useful output (e.g., protein fitness) is known as "semi-supervised learning." This name comes from the fact that the process consists of both an unsupervised learning stage (that is, training an ML model on unlabeled data—for instance, on unlabeled protein sequences) and a supervised stage (training an ML model on labeled data). The details of how exactly the unsupervised stage (where a model is trained on unlabeled protein data to produce embeddings) is performed are complicated and vary by strategy.[49,51,60,61,52–59] Indeed, how to best perform this stage of the semi-supervised pipeline—and even whether the learned embeddings produced by existing strategies and models provide a benefit over simpler encoding strategies—is still largely debated,[33,62–66] and is the focus of the work I present in Chapter 3.

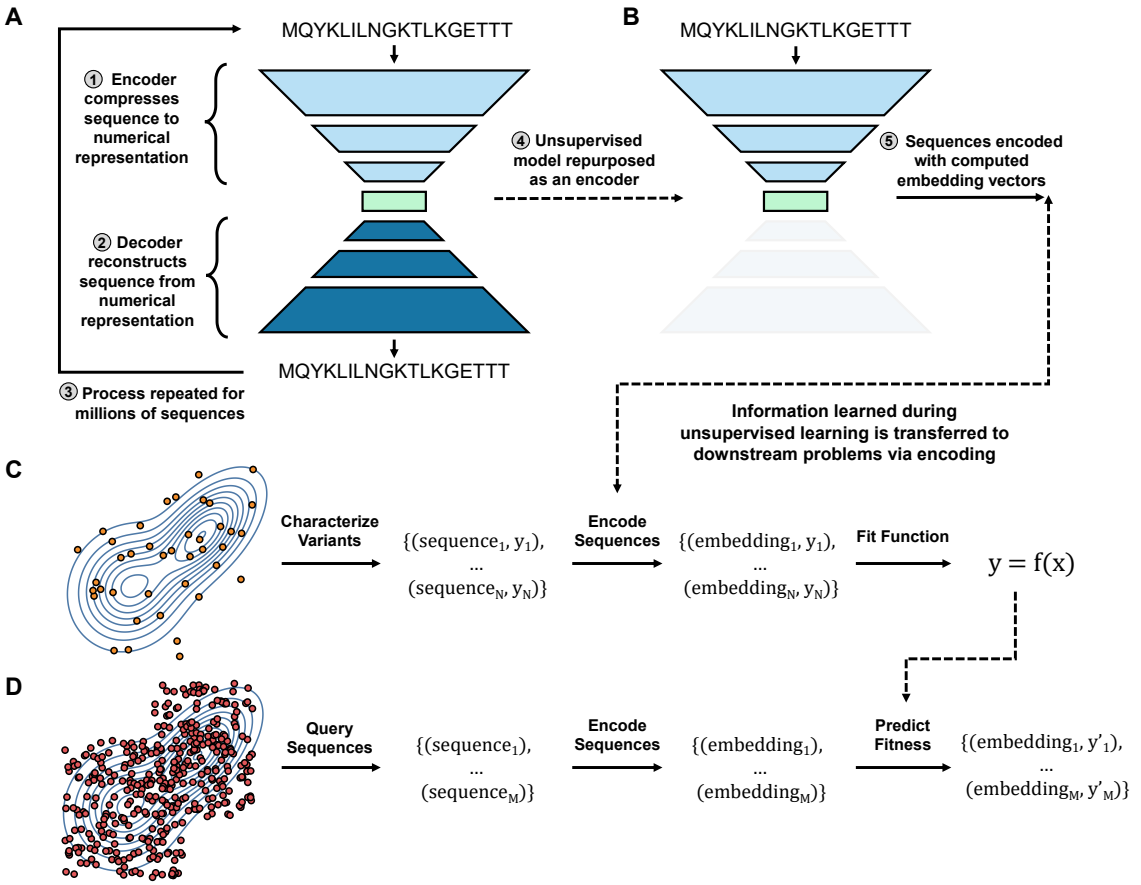


Figure 1-2. An example semi-supervised learning workflow illustrated using an autoencoder as the unsupervised model. (A) In this example, during the unsupervised stage, an autoencoder is trained to compress ("encode") protein sequences to a numerical representation and then use that representation to reconstruct ("decode") the sequences. The compression during encoding creates an information bottleneck (the central green layer in the figure) that forces the model to extract the most relevant features of protein sequences; the more informative the extracted features, the greater the model's ability to reconstruct sequences. (B) Once the unsupervised model is trained, the protein sequence encoder may be repurposed by removing the decoder module and taking the bottleneck ("embedding") layer as an encoding. This encoding transfers information learned during unsupervised training to a supervised process, in principle decreasing the amount of required labeled data. (C) During supervised training, an additional "top" model is trained to relate the encoded sequences to their characterized fitness values. The parameters defining the encoder can either be frozen (i.e., the encoder is not modified during supervised training) or further fine-tuned (i.e., the encoder is further trained along with the top model for the specific supervised task) during supervised training. (D) As more sequences are drawn from the fitness landscape, they are first encoded by the encoder, then passed into the learned function to predict the fitness of previously unseen protein variants.

1.2.3 Prediction of Protein Fitness with Models Trained on Sequence Data

So far, I have presented the primary goal of MLPE as using labeled sequence-fitness training data to build the function *f(sequence) = fitness*. However, this is not the only way that MLPE can be performed. Because the goal of MLPE is to identify the highest-fitness proteins, we do not need to be able to *precisely* predict the fitness of new protein variants. Indeed, we only need to be able to predict the *relative* fitnesses of different protein variants, and could therefore perform MLPE just as effectively using a function of the form rank(*f(sequence)*) = rank(*fitness*), where we are not predicting fitness itself, but instead something correlated to it.

As discussed in the previous section, all proteins found in modern sequence databases must follow some set of biophysical and evolutionary rules that allow them to perform a useful biological function, and we can train models to learn the sequence constraints that result from those rules. While one use of these models is to produce protein embeddings, many of them can also be used to make what are known as "zero-shot" predictions of protein fitness, which is defined as "prediction of protein fitness without the need for collection of additional labeled data."[33,67,68] Importantly, zero-shot predictors do not output a prediction of protein fitness directly but instead some score that is expected to be correlated with it.[d] The effectiveness of a zero-shot predictor is thus determined by the degree of correlation between its proposed score and true fitness, where the ideal predictor would generate scores perfectly proportional to fitness. While, in principle, any source of data that is believed to contain information correlated with protein fitness can be used to build a zero-shot predictor,[69] much recent effort has focused on the development of sequence-based zero-shot predictors.[33,57,67,68,70]

In some capacity, any model used for sequence-based zero-shot prediction can be thought of as a "generative model," which is one trained to learn a representation or approximation of the underlying distribution of a dataset. Many of the models used for building protein

[d] As mentioned in an earlier footnote, predicting scores correlated with fitness is typical for rational design strategies. Indeed, as discussed in detail in Chapter 2, models used by rational design can be considered to be (and effectively used as) non-ML zero-shot predictors.

embeddings discussed in the previous section can also be considered generative models and can be used for both embedding production and zero-shot prediction.[55,57,67,68] When these models are trained on unlabeled protein sequence data, they learn a representation of the distribution of allowed protein sequences that captures the rules governing sequence constraints (Figure 1-3A). When these models are used to encode proteins, those rules are assumed to be represented in the derived embeddings, thus creating a more informative encoding that can be passed into a downstream supervised task (Figure 1-2);[64] when these models are used for zero-shot prediction, they are used to score candidate proteins based on how well they obey the learned rules, with candidates that better obey assumed to be more likely to show some degree of a desired fitness than those that do not. From a more probabilistic point of view, we can think of scoring proteins in this way as querying the likelihood that a new protein sequence was generated from the learned distribution of underlying sequences (Figure 1-3B, C). If a sequence is highly likely to belong to the learned distribution, then it is more likely to be a functional protein, and vice versa. Of course, all of this assumes that the target fitness to be engineered correlates well with evolutionarily optimized fitness, but this will not always be the case (Figure 1-3A). As stated earlier, the quality of a zero-shot predictor depends entirely on how well its output score correlates with the target fitness.

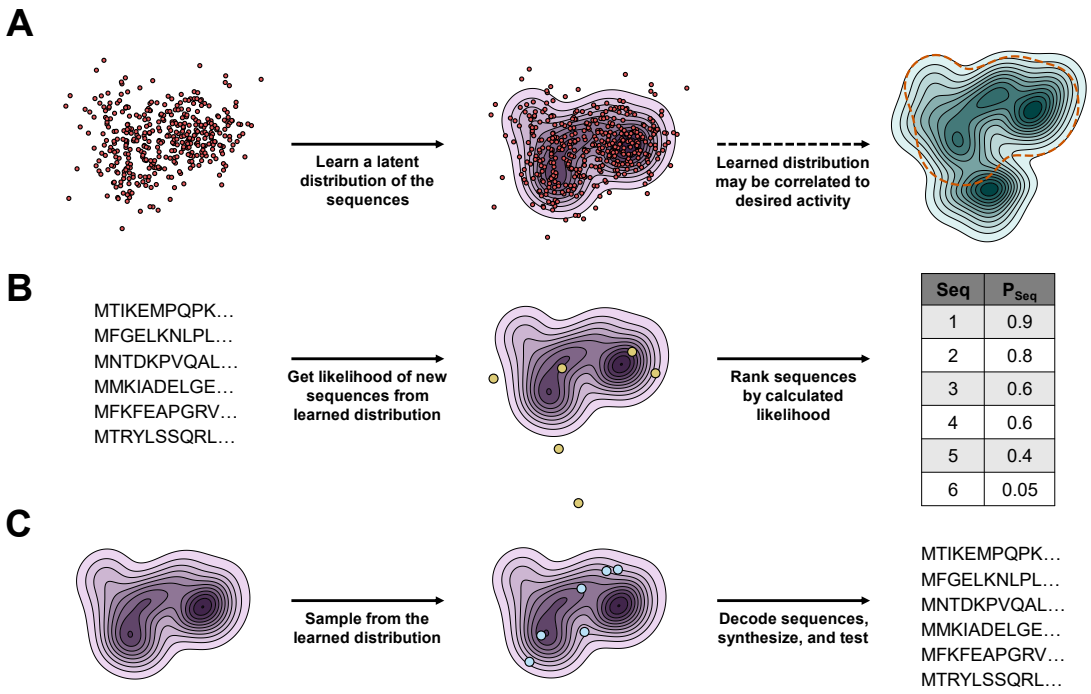


Figure 1-3. An illustration of the use of generative models for zero-shot prediction and sequence generation. (A) Generative models learn a representation of the distribution of allowed protein sequences from those used to train them. This distribution can correlate with the fitness landscape (green) for a desired activity, but the two distributions may not necessarily overlap. (B) When generative models are used for zero-shot prediction, it is assumed that the learned distribution correlates well with the distribution of target activity. When used in this capacity, the model is used to find the likelihood for new sequences. Sequences with high likelihood can be prioritized for screening under the assumption that a higher likelihood corresponds to higher fitness or at least higher probability of maintaining some degree of function. (C) When generative models are used for sequence generation, new sequences are drawn from the learned distribution. Sequences with higher likelihood are more likely to be drawn, so the drawn sequences tend to be functionally similar to those used for training.

1.2.4 Machine Learning-Guided Navigation of Protein Fitness Landscapes

With a trained model in hand—whether built using unlabeled or labeled data—the protein fitness landscape can now be explored computationally to find new, improved proteins. If exploring narrow regions of sequence space (e.g., a combinatorial landscape built by mutating no more than five or six positions simultaneously), then this is a simple task: the model can be used to enumerate the space, predicting the fitness (or a correlated score) for all possibilities. Once larger regions of sequence space are considered, however, this “brute

force" approach fails, as even though computers can evaluate protein candidates at a rate orders-of-magnitude faster than laboratory screening, they are still far too slow for comprehensive exploration of even moderately sized regions of sequence space.[e]

To keep computational screening burdens low, non-brute force ML-guided exploration of large fitness landscapes typically proceeds through rounds of (1) proposing a library of new sequences and then (2) using an ML model to computationally evaluate those sequences and identify improved ones.[23,71] As in directed evolution, screening burdens are lowered by iteratively climbing toward improved regions of the fitness landscape rather than trying to exhaustively explore it. Unlike directed evolution, however, because these strategies are run in a computer, their screening capacity is much higher, allowing for exploration of much larger fitness landscapes.

The various strategies for accomplishing iterative ML-guided exploration of fitness landscapes have been covered in recent literature,[23,71] so I will not go into detail here, though it suffices to say that their main differences are in how they propose the next batch of sequences to evaluate. For example, some strategies are direct computational analogues of evolution, iteratively building mutant protein sequences *in silico*, feeding them into a trained ML model to assign them a score, then using the most improved mutant (or mutants) as the starting point for the next round of computational mutagenesis and evaluation.[64,72,73] Other strategies instead rely on generative models to propose new batches of proteins, iteratively drawing new sequences from the learned distribution of special classes of generative models (much like we would draw samples from a normal, binomial, or any other probability distribution), using a model to assign those sequences a score, then using the sequences with the highest predicted scores to update the generative model, thus biasing it toward producing higher fitness variants in the next round.[74,75] Yet more strategies directly optimize protein fitness in a learned embedding space, iteratively proposing new batches of embedding

[e] For instance, say we could evaluate 1 billion proteins per second, a fast rate even for a computer. If we were to consider just 20 positions at once in a combinatorial library, it would take around 3.3 billion years to evaluate all combinations, which is around a quarter the age of the universe.

vectors to be passed into a model and scored, then decoding the optimal vectors to identify an optimal protein once a sufficiently fit one has been identified.[76,77]

Regardless of the ML-guided fitness landscape navigation strategy used, all have the same goal: to maximize the breadth of sequence space that can be explored while minimizing the need for experimental evaluation of proteins. Despite continued advances in our ability to train MLPE models with less data, however,[33,64] none of these strategies will ever completely eliminate the need for experimental evaluation of proteins. For one, at the end of computational exploration, the best predicted sequences will need to be experimentally verified; after all, the goal is not to predict fitness, but to identify sequences whose true fitness is optimal. Perhaps more importantly, however, as computational exploration proceeds further and further away from the region of sequence space occupied by the initial training data, the quality of model predictions will deteriorate, necessitating occasional collection of additional training data.

Recall from Section 1.2.1 that the accuracy of an ML model is determined in large part by the data used to train it. Specifically, the accuracy of the ML model will deteriorate when it is used to make predictions in regions of the input space not captured by its training data, a situation known as "dataset shift." Exploration of protein fitness landscapes inherently requires a degree of dataset shift, however, as optimized proteins do not exist in the region of sequence space used for training data; otherwise, we would not be performing optimization in the first place. As exploration takes a model further from the initial training data, the degree of dataset shift will increase and the model's predictions will steadily deteriorate, eventually to the point where they are no longer useful and the model must be updated with newly collected training data. Of course, because the goal of MLPE is to both identify improved proteins *and* reduce experimental screening burden, this sets up a tradeoff: there is a balance to be found between how much we *exploit* knowledge from existing training data—limiting our search to regions of the fitness landscape where the model is believed to still make reasonably accurate predictions—and how much we *explore* new

regions of a fitness landscape—necessitating collection of new training data but also potentially identifying better proteins than we would have otherwise found.

Unfortunately, the ideal solution to this "exploration-exploitation tradeoff" is rarely clear, as it depends on factors ranging from the shape of the ground truth fitness landscape to experimental screening capacity, the specific goals of the experiment, and the model class used for modeling. It is easier to find a balance with some model classes than others, however. Gaussian processes, for instance, are among the most popular models used in ML-guided exploration and will return predictions along with an uncertainty value that is correlated with the distance from the points in the training data.[78,79] As predictions are made further from the training data, the level of uncertainty goes up, providing information that can be used to identify both those model predictions that would require collection of more training data to be confirmed (exploration) and those that can be immediately trusted based on existing data (exploitation). Most other model designs commonly used in MLPE, however, do not provide uncertainty estimates, making it challenging to identify how far away from the initial training set they can be confidently used. In these cases, a common and simple practice is to set a "trust radius," where model predictions are no longer trusted when used to score protein sequences some number of mutations away from the original (un-engineered) protein sequence.[64,73] More statistically grounded methods have been proposed as alternates and typically involve either weighting the probability that a protein sequence will be proposed for evaluation by its similarity to the distribution of training data or weighting the fitness score assigned to an evaluated protein sequence by its similarity to the distribution of training data.[75,80] In general, the development of optimal strategies for (1) building models that can make robust predictions further from training data and (2) bounding exploration of input space to regions where models remain trustworthy remains an active area of research both within and outside of MLPE.

1.3 Thoughts on Ongoing Challenges for Machine Learning-Assisted Protein Engineering

Section 1.2 covered the basic steps in the optimization stage of a typical MLPE pipeline, including recent advances in strategies for each of them. In this section, I will focus instead on the major challenges currently facing MLPE, regardless of the stage of the pipeline. In my view, if these challenges were overcome, they would represent large leaps forward for both MLPE and protein engineering as a whole.

1.3.1 Practical Considerations for the Wet Lab Application of MLPE

MLPE is a hybrid wet lab-computational strategy—at some point, someone must physically perform the experiments to gather training data and evaluate predictions. Throughout this chapter, however, I have so far primarily focused on its computational component. Indeed, as a community, MLPE researchers often tend to neglect the wet lab considerations, essentially assuming that the computational and wet lab components can be decoupled. This is, unfortunately, a false assumption, and the wet lab component of MLPE places significant restrictions on what can be considered a practically applicable—or at the very least a financially competitive—MLPE approach.

The most obvious wet lab restriction is the limited ability to screen protein candidates, which constrains the availability of the labeled sequence-fitness data needed to train the most effective MLPE models. Indeed, MLPE researchers are most cognizant of this wet-lab limitation, and it is the core driving factor for the development of the zero-shot and semi-supervised methods discussed in Sections 1.2.2 and 1.2.3; it is also the driving factor behind my development of the informed training set selection strategies discussed in Chapter 2. Our restricted ability to collect labeled sequence-fitness data is not only caused by bounded screening capacity, however; it is also caused by the cost of *sequencing* protein variants.

One of the greatest strengths of directed evolution is the simplicity with which it can be performed. Among the many traits that make it such a simple process is its lack of requirement for sequencing. Indeed, a directed evolution endeavor could be completed without ever sequencing a single protein, though in practice a few protein variants will

typically be sequenced (purely for informational purposes) during each iteration of mutagenesis/screening. This stands in contrast to MLPE methods, which require sequencing at least tens but usually hundreds or thousands of variants to build a sufficiently sized labeled training dataset. Using the standard sequencing strategy employed by directed evolution—Sanger sequencing—the cost of gathering labeled training data can easily balloon into the thousands of dollars, as I saw in the first project to which I contributed at Caltech.[81,82] Sequencing requirements thus place an additional effort and financial burden on protein engineers using MLPE over directed evolution, reducing the practical utility of MLPE strategies.

As part of my thesis work, I developed a suite of tools that eliminates the sequencing burden of MLPE, at least when building training sets from specific protein library designs. Detailed in Chapter 4, this toolset employs next generation sequencing (NGS) technology—an alternate sequencing strategy to Sanger—to drop the cost and effort of sequencing the hundreds of variants typically needed to build effective MLPE training datasets down to a level similar to the cost of sequencing the few variants typically sequenced per round of directed evolution.[83,84] This technology should generally make MLPE more practically applicable; it should also enable the construction of a large database of protein sequence-fitness data, the utility of which is discussed in Sections 1.3.3 and 1.3.4. Unfortunately, however, the cost of sequencing is not the only wet-lab component that limits the application of MLPE.

The models used in MLPE are rarely accurate enough for an engineer to have the highest confidence in their predictions, so typically a set of the predicted-best protein variants are built and evaluated. This is done to improve the probability that at least one (but ideally multiple) of those variants satisfies the engineering goal. Cost-effectively building these designs can be extremely difficult, however, presenting another wet-lab limitation of MLPE.

Regardless of protein engineering application, when many protein variants must be built, they are typically not constructed individually. Instead, to keep costs low, they are constructed in the same test tube in a highly multiplexed fashion. I will not go into the details

of methods for library construction here, leaving that for the referenced sources;[11,30] it is important to note, however, that a critical component to many of these methods is "degenerate oligonucleotides." A degenerate oligonucleotide consists of a pool of DNA oligonucleotides (short strands of DNA) that have randomized bases at at least some positions. For example, a very simple degenerate oligonucleotide might contain the sequences ATGGGC and ATGCGA—positions 4 and 6 have randomized bases in this example. Importantly, randomization does not need to include all four possible bases, so the DNA sequences in the degenerate oligonucleotide can be restricted to encode a specific set of protein variants. Unfortunately, however, it is not possible to decouple randomization at different positions in the most cost-effective methods for degenerate oligonucleotide production; thus, any protein variant library that must be constructed using randomization at multiple DNA positions will invariably also include off-target designs.[f]

To build a specified library of protein variants, an ideal degenerate oligonucleotide that minimizes off-target designs and maximizes on-target designs (while keeping them as uniformly represented as possible) must first be identified. This is done by taking advantage of the redundancy of the genetic code—multiple different DNA sequences can encode the same protein sequence, and depending on the set of protein sequences desired, certain DNA sequences will produce fewer off-target variants and better balance the representation of on-target designs in the final library than others.

While simple on paper, it is often extremely computationally intensive to identify an appropriate degenerate oligonucleotide for building a specified protein variant library, if one can be found at all. For directed evolution, this is typically not a challenge. In most applications, the target library is the same—one that uniformly represents all 20 amino acids at a given position in a protein—and a set of optimized degenerate oligonucleotide designs that can build it has already been identified.[85] For MLPE, this is more problematic, as a new

[f] For instance, say I wanted to include the DNA fragments ATG and CTA in my degenerate oligonucleotide. I would thus want to randomize the first position to include "A" and "C" and the last position to include "G" and "A." Because randomization at different positions cannot be decoupled, however, in addition to the desired fragments, the final degenerate oligonucleotide would also include ATA and CTG in it.

degenerate oligonucleotide must be designed for each library of predicted-best variants. Over the years, increasingly more effective tools have been developed for performing this search;[86–90] however, even when these tools can complete the search in a reasonable amount of time, there is no guarantee that they will find an appropriate design. Oftentimes, using traditional degenerate oligonucleotide construction, it is not possible to actually build designs predicted to be optimal by MLPE models.

Promisingly, in recent years, new approaches to DNA synthesis have made it possible to build large, user-specified populations of oligonucleotides, eliminating the major limitations of traditional degenerate oligonucleotide design. Twist Bioscience, for instance, offers an "Oligo Pool" service that allows users to order pools of DNA oligonucleotides containing thousands of uniquely specified DNA sequences. With the ability to specify unique DNA sequences, precise libraries of proteins—uniformly represented and containing no off-targets—can now be constructed. Unfortunately, while strategies like Twist Bioscience's Oligo Pools have already been put to excellent use,[91] they can currently be prohibitively expensive for regular incorporation in the MLPE workflows of many groups, particularly those in academia. Until the cost of these new strategies is lowered—and even after the fact, for that matter—there may be value in designing strategies that explicitly consider both MLPE model predictions and the practical limitations imposed by degenerate oligonucleotide design when choosing the proteins to construct.

1.3.2 Transfer Learning

We generally expect that data directly related to the problem we are trying to solve will be most valuable. For instance, if I am trying to build an MLPE model to predict some fitness, then the most valuable data would be a large dataset relating protein sequence to that fitness. Given the beyond-astronomical size of possible protein space, however, we will always be working in a comparatively data-limited setting, so it would be valuable to develop strategies that can extract helpful information from other sources of data as well.

To date, most efforts to develop transfer learning strategies for MLPE have focused on extracting information from sequences, largely because protein sequences are the most

abundant and easily accessible source of protein data. As introduced in Section 1.2.2, some of these approaches attempt to augment small, labeled datasets by building informative embeddings from protein sequence data. As introduced in Section 1.2.3, other approaches attempt to eliminate the need for labeled data entirely, performing zero-shot predictions of protein fitness using information extracted from sequence data alone. Of these two general strategies, zero-shot predictors have so far proved the most useful for protein fitness predictions, though learned embeddings have proven useful in a number of other protein-related tasks.[92–98]

Exactly why learned protein embeddings are not currently so helpful for protein fitness prediction is unclear. Early studies using learned embeddings convincingly suggested that learned embeddings can reduce our need for labeled data;[64] however, the initial excitement faded once later studies showed that, in many instances, uninformative encoding strategies like one-hot can be just as effective as learned embeddings.[62,65] While these initial reports suggested that learned embeddings can still be more useful than one-hot in extremely labeled data-limited settings, I have found in my own work (as shall be shown in Chapter 2) that this is not always the case.[33]

To a degree, some of the variability in the conclusions about the effectiveness of learned embeddings for protein fitness prediction will come from the variability in the protein fitness landscapes to which they are applied. Our protein sequence databases are biased toward the representation of certain protein families over others; we thus might expect the embedding quality of well-represented proteins to be higher than that of underrepresented ones. It must also be noted that most of the models currently used for building learned protein embeddings are directly adapted from natural language processing (NLP). While there are certainly some similarities between protein sequences and written language, there are also some major differences.[99] Therefore, we might also expect that models designed to more explicitly capture the idiosyncrasies of proteins would be more effective. Indeed, Chapter 3 is focused on my efforts to build and evaluate the effectiveness of such a model.

1.3.3 The Starting Point Problem

So far, I have primarily discussed MLPE from the perspective of protein fitness optimization. Before optimization can be performed, however, a protein with some degree of the desired function—a starting point—must be identified. Unfortunately, solving this "starting point problem" is extremely difficult and is the point where most protein engineering projects fail, regardless of any engineering strategy that would be used.

As with optimization approaches, starting point identification strategies must contend with the challenges imposed by the beyond-astronomical size of protein sequence space and our limited ability to experimentally search it. Generally, any strategy that aims to search sequence space must have a way to restrict its search to keep screening burdens practical. Optimization strategies like directed evolution and ML-guided navigation accomplish this by relying on the fact that functional proteins tend to be clustered together in sequence space; thus, they only evaluate protein candidates that are close in sequence space to a known functional example (or examples). The same approach cannot be taken for finding a starting point, as by definition there are no known proteins with the desired function. Instead, the most common approach is to test existing proteins with a related function for some degree of promiscuous activity for the desired one, now relying on the fact that proteins that perform similar functions often have similar sequences.

Exactly how candidate promiscuous proteins are identified varies, but they are most commonly sourced either from databases of known functions or past engineering projects.[14] Importantly, regardless of candidate source, only *existing* proteins are typically searched—it is very rare for mutants of candidates to be produced and evaluated when looking for a starting point. At first, this may seem an odd restriction, as it is not uncommon for new promiscuous activities to emerge in protein lineages while they are engineered for different tasks and it may indeed be that a few mutations to an existing protein would yield the desired activity.[100,101] However, even considering one or two mutations to a set of candidates would dramatically increase the screening burden for a search, and even though it is *possible* that mutants would exhibit the desired activity, chances are still high that if a given protein does

not perform the desired function, a protein a few mutations away will not perform it either. Given a limited screening budget, it is thus generally regarded as a waste of laboratory resources to build and screen mutants of existing proteins.

Because computational approaches expand our screening capacity, it seems practical that they could be used to screen for starting points *in silico*, allowing for evaluation of mutants of existing candidates. Indeed, it is not wholly uncommon to use rational design strategies to screen candidate proteins for a new function, though in most of these cases the target function is some form of binding (either to another protein or a small molecule). This is largely because binding is among the simplest functions to understand and model; binding functions also tend to inhabit wider windows of sequence space than others and can often be found even when using somewhat imperfect models to search. Comparatively, it is far more difficult to use rational design to find a starting point for more complicated functions that arise from more intricate and delicate physical and chemical mechanisms (for instance, enzyme catalysis), as even the slightest error in the model can result in erroneous predictions. For functions where we have no understanding of the underlying mechanism at all, rational design becomes impossible to use for starting point identification.

Because ML relies on data, not physical understanding, we might expect it to be more effective than rational design as a more general-purpose starting point-identification approach. To build an MLPE model that is accurate enough for starting point identification, extensive sequence-function data must be available, and as the complexity of the mechanism underlying the desired function increases, the amount of data needed will increase as well. Unfortunately, for many functions there is currently not enough data to build effective models, and so, just like rational design, MLPE methods for starting point identification are primarily limited to binding.[16,79,102–106] There are exceptions, however, and as we might expect they are found primarily in protein families that have been extensively studied and for which large databases of sequence-function data already exist.[107,108]

As already discussed in Section 1.3.1, it has historically been too expensive to gather extensive sequence-fitness datasets, and this largely explains why we do not have the data

available to train starting point predictors for most protein families and functions. However, cost-effective data collection strategies like evSeq and DMS (discussed in Section 1.3.4) promise to change this, and it is reasonable to assume that we will see far more sequence-function data made available in the future, better enabling MLPE-based starting point identification.

1.3.4 Data Availability and Benchmarking for Protein Fitness Prediction

Large, well curated, publicly available datasets have historically proven critical to the advancement of ML applications, regardless of field. For instance, within the broader ML community, the database of human-annotated images "ImageNet" is typically credited with ushering in a period of rapid advancement in computer vision.[109] Likewise within the protein engineering community, the recent efforts in protein sequence-based semi-supervised, representation, and transfer learning discussed throughout this chapter were only made possible by the availability of large protein sequence databases like UniProt, Uniclust, and UniRef.[22,37–39,41,42,110] Similarly, the recent dramatic advances in ML-based protein structure prediction with AlphaFold and RoseTTAFold were largely enabled by the information contained in both sequence databases and protein structure databases like the PDB and CATH.[41,42,95–97,111,112] It stands to reason, then, that a suitably large, organized database of protein sequence-fitness information would spawn an era of rapid advancement in protein fitness prediction, be that for optimization or solving the starting point problem.

Unfortunately, there is currently no large, well-maintained database of protein sequence-fitness information, though the framework for one does exist.[113] As is explained in Chapter 4, this is largely because it has historically been prohibitively expensive to gather sequence-fitness data. Indeed, those large sequence-fitness datasets that *are* currently available almost exclusively come from experiments using a uniquely cost-effective strategy known as "deep mutational scanning" (DMS), which can be used to generate sequence-fitness data for hundreds of thousands of protein variants at a total cost of hundreds to thousands of dollars.[114,115] DMS can only be applied for very specific definitions of fitness, however, and

while the datasets produced by it have proven essential for advancing MLPE, they capture a very limited range of protein scaffolds and activities.

In Chapter 4, I discuss a new method that I developed called "every variant sequencing" (evSeq) that, unlike DMS, can be used to cost-effectively generate sequence-fitness data regardless of the definition of fitness. It was designed to be incorporated into existing directed evolution workflows, thus allowing the sequence-fitness information for all protein variants produced during an engineering endeavor to be recorded, which is something that is not currently done. Widespread adoption of evSeq across the protein engineering community has the potential to produce hundreds of thousands—if not millions—of sequence-fitness datapoints per year covering a wide range of protein scaffolds and effectively any definition of fitness that can be imagined.[g] Such volume of data would allow us to rapidly build a sequence-fitness database that could be used to support, among many other things, the further development of MLPE.

Importantly, while a database on its own *can* be enough to advance an ML application, progress can be further expedited by the availability of well-designed benchmark tasks. A benchmark task is one designed to represent a typical application, providing a metric for how well we might expect an ML strategy to perform in the real world; they are designed such that newly proposed model architectures, training procedures, etc., can be compared in a fair, standardized manner. This standardization effectively enables community-wide collaboration toward solving a common problem. It is thus unsurprising that well-designed benchmark tasks are often central to some of the most decisive advances in ML. Returning to the earlier example of ImageNet, for example, advances in computer vision did not arise due to the publication of that dataset alone; the associated competition "ImageNet Large Scale Visual Recognition Challenge" (ILSVRC) annually provided a new set of publicly available benchmark tasks. Similarly, advances in ML-guided protein structure prediction

[g] In the Arnold lab, we on average complete around 10 directed evolution endeavors per year. In each of these projects, we will screen hundreds to thousands of protein variants for fitness. If we assume that the thousands of other research groups performing directed evolution complete projects at a similar rate and scale, then as a community we can produce somewhere between 1,000,000 and 10,000,000 sequence-fitness datapoints per year.

did not arise solely due to the availability of large sequence and structure databases; the biennial competition "Critical Assessment of Structure Prediction" (CASP) provided (and continues to provide) standardized protein structure modeling benchmark tasks for over 25 years before the breakthroughs that were AlphaFold and RoseTTAFold.[116]

To date, there is no ILSVRC- or CASP-type competition for protein fitness prediction, though efforts have been made to design effective benchmarks. Somewhat unofficially, a series of DMS datasets has found itself commonly used to benchmark, in particular, new zero-shot prediction methods.[67,68,70,117] More concerted efforts to create standardized benchmarks exist, however, with "Tasks Assessing Protein Embeddings" (TAPE) providing one example.[58] The benchmark tasks in TAPE were created to assist in the development of new protein encoding schemes and cover a variety of different ML applications to proteins, two of which are fitness optimization tasks. The ideal ML strategy for fitness prediction is likely to vary by fitness landscape and application, however, so these two tasks are certainly not enough to robustly benchmark new fitness optimization approaches. To address this problem, I contributed to the development of the "Fitness Landscape Inference for Proteins" (FLIP) set of benchmark tasks.[118] The fifteen FLIP tasks rely on three protein fitness landscapes of varied structure and, importantly, are inspired by a variety of real-world protein engineering applications, thus encouraging the community to create MLPE methods that are generalizable to a number of different use cases.

The benchmark tasks in FLIP and TAPE are an excellent start, but they are highly unlikely to have the same impact as ILSVRC and CASP. Those two competitions have seen sustained, organized, community-wide investment for multiple years (multiple decades in the case of CASP), something that neither TAPE nor FLIP will provide. However, with new tools coming online for the high-throughput production of sequence-fitness data and ever-growing interest in MLPE, perhaps a similar competition can be built for fitness prediction tasks. Such an approach would almost certainly lead to rapid advancement of applicable MLPE models and strategies.

Chapter 2

INFORMED TRAINING SET DESIGN ENABLES EFFICIENT MACHINE LEARNING-ASSISTED DIRECTED PROTEIN EVOLUTION

Material from this chapter appears in **Wittmann, B. J.**; Yue, Y.; and Arnold, F. H. (2021) Informed Training Set Design Enables Efficient Machine Learning-Assisted Directed Protein Evolution. *Cell Syst. 12*, 1026-1045.e7. https://doi.org/10.1016/j.cels.2021.07.008.

Abstract

Directed evolution of proteins often involves a greedy optimization in which the mutation in the highest-fitness variant identified in each round of single-site mutagenesis is fixed. The efficiency of such a single-step greedy walk depends on the order in which beneficial mutations are identified—the process is path-dependent. Here, I investigate and optimize a path-independent machine learning-assisted directed evolution (MLDE) protocol that allows *in silico* screening of full combinatorial libraries. In particular, I evaluate the importance of different protein encoding strategies, training procedures, models, and training set design strategies on MLDE outcome, finding the most important consideration to be the implementation of strategies that reduce inclusion of minimally informative "holes" (protein variants with zero or extremely low fitness) in training data. When applied to an epistatic, hole-filled, four-site combinatorial fitness landscape, my optimized protocol achieved the global fitness maximum up to 81-fold more frequently than single-step greedy optimization.

2.1 Introduction for Chapter 2

Enzyme engineering has revolutionized multiple industries by making chemical processes cheaper, greener, less wasteful, and overall more efficient. Enzymes and other proteins are engineered by searching the protein fitness landscape, a surface in a high-dimensional space that relates a desired function ("fitness") to amino acid sequences.[27,29] Exploring this landscape is extremely challenging: the search space grows exponentially with the number of amino acid positions considered, functional proteins are extremely rare, and experimental screening of proteins can be resource-intensive, with researchers often limited to testing a few hundred or thousand variants. Directed evolution (DE) can overcome these challenges by employing a greedy local search to optimize protein fitness.[10] In its lowest-screening-burden form (hereafter referred to as "traditional DE"), DE starts from a protein having some level of the desired function, then iterates through rounds of mutagenesis and screening, where in each round single mutations are made (e.g., by site saturation mutagenesis) to create a library of variants and the best variant is identified and fixed; iteration continues until a suitable level of improvement is achieved (Figure 2-1A).

By focusing on single mutations rather than combinations of mutations, traditional DE can be used to optimize protein fitness with a low screening burden. The process is highly effective when the beneficial effects of mutations made at different sequence positions are additive; however, focusing on single mutants ignores the codependence of mutations (epistasis).[119,120] Epistasis is commonly observed, for example, between residues close together in an enzyme active site or protein binding pocket, where mutations often affect function. Epistatic effects can decrease the efficiency of DE by altering the shape of the protein fitness landscape. Specifically, epistasis can alter gradients on the fitness landscape to make the route to a global optimum very long,[121] or it can introduce local optima at which traditional DE can become trapped (Figure 2-1B). Both lower the average fitness that can be achieved for a given screening burden. The only way to account for epistasis during optimization is to evaluate and fix combinations of mutations, bypassing the path-dependence of traditional DE. Due to limited screening capacity, however, this is intractable for most protein engineering projects.

Increasingly, machine learning (ML) is being used to ease experimental screening burden by evaluating proteins *in silico*.[15–19,22,37,71,110] Data-driven ML models learn a function that approximates the protein fitness landscape, and they require little to no physical, chemical, or biological knowledge of the problem. Once trained, these models are used to predict the fitness of previously unseen protein variants, dramatically increasing screening capacity and expanding the scope of the protein fitness landscape that can be explored by replacing expensive laboratory experimentation with *in silico* screening. The first project to which I contributed at Caltech demonstrated a machine learning-assisted directed evolution (MLDE) strategy for navigating epistatic fitness landscapes that cover a small number of amino acid sites.[82] MLDE works by training an ML model on a small sample (10^1–10^2) of variants from a multi-site simultaneous saturation mutagenesis ("combinatorial") library, each with an experimentally determined fitness (i.e., a model is trained using a small sample of sequence data labeled by fitness); the model is then used to predict the fitnesses of all remaining variants in the combinatorial library (10^4–10^5), effectively exploring the full combinatorial space. Combinations with the highest predicted fitness are experimentally evaluated, the best combination is fixed, and another round of MLDE is started at a new set of positions (Figure 2-1C). The iterative nature of MLDE is identical to that of traditional DE, but by evaluating and fixing multiple cooperative mutations, MLDE avoids some local fitness traps or long paths to the global optimum for each combinatorial library.

The original MLDE work serves as a baseline, as it did not explore the many design considerations of MLDE (Figure 2-1C, bold and underlined questions).[82] Two notable considerations are (1) the choice of encoding strategy and (2) the handling of low-fitness variants in combinatorial libraries. Protein sequences must be numerically encoded to be used in ML algorithms, and the choice of encoding will affect the outcome of learning. In the original implementation, a one-hot encoding scheme was used, which is a simple categorical encoding that captures no information about the biochemical similarities and differences of amino acids. Mutating an amino acid to a similar one (in terms of size, charge, etc.) is less likely to affect protein fitness than mutating it to a very different one, however, and this knowledge can be transferred into ML models via the encoding strategy. The

effectiveness of an ML model is also determined by the information content of the data used to train it, and so the choice of variants to use for the training stage of MLDE is important. Combinatorial libraries tend to be enriched in zero- or extremely low-fitness variants, particularly when constructed in regions critical to protein function like an enzyme active site.[28,78,122] These "holes" provide minimal information about the topology of the regions of interest in a fitness landscape (i.e., they provide no information about regions with functioning proteins and no information about the *extent* to which different mutations affect fitness) and can bias ML models to be more effective at predicting low-fitness variants than high-fitness ones, the opposite of the goal. In the original MLDE implementation, high-sequence-diversity training data was generated by sampling randomly from full combinatorial spaces. Because combinatorial landscapes tend to be dominated by holes, however, this random draw primarily returned sequences with extremely low or zero fitness, resulting in training data that, despite containing diverse sequences, was information poor.

In this chapter, I evaluate various design considerations by simulating MLDE on the empirically determined four-site combinatorial fitness landscape (total theoretical size of 20^4 = 160,000 protein variants) of protein G domain B1 (GB1) (Figure 2-1D).[123] This landscape contains multiple fitness peaks (the routes to which are not always direct) and is heavily populated by zero- and low-fitness variants (92% have fitness below 1% of that of the global maximum), and thus not only presents an ideal testing ground in which to compare the abilities of traditional DE and MLDE to navigate epistatic fitness landscapes, but also serves to test the ability of ML methods to navigate hole-filled regions of protein fitness landscapes. I begin by evaluating a number of alternate encoding strategies to one-hot, including physicochemical encodings (encodings that capture physical and chemical properties of different amino acids) and learned embeddings derived from eight different natural language processing models (which are encodings extracted from machine learning models that represent physicochemical and contextual information about different amino acids—more background is provided in the relevant section).[51,55,58,61,124] Next, I demonstrate how integration of models and training procedures better tailored for protein fitness landscapes into the workflow can improve MLDE performance.[125–127] I then show the importance of

reducing uninformative holes in MLDE training sets and propose integrating a form of zero-shot prediction (i.e., prediction of variant fitness prior to data collection) into the MLDE pipeline to generate more informative training data. I call the general strategy of running MLDE with training sets designed to avoid holes "focused training MLDE" (ftMLDE). I next evaluate the effectiveness of a number of zero-shot strategies for designing training data for ftMLDE applied to GB1, including state-of-the-art strategies that leverage local evolutionary information from multiple sequence alignments (MSAs),[68,70] a "masked-token prediction" strategy that leverages global sequence information derived from large sequence databases,[51,55,61,128] and predicted ΔΔG of protein stability upon mutation. I then use the effective zero-shot predictors to generate information-rich training data. Finally, using this training data, I test the effect of training set size, the zero-shot predictor used for training set construction, and protein encoding on the outcome of ftMLDE.

In all, I found that, while using more informative encodings and models better-tailored for combinatorial fitness landscapes could improve MLDE outcome, the most important design consideration was training set design, with ftMLDE generally showing improved identification of the GB1 global fitness maximum compared to MLDE. My most effective combination of MLDE design considerations—384 training points chosen using predicted ΔΔG as the zero-shot predictor and with sequences encoded using embeddings derived from the recently published MSA Transformer—successfully identified the GB1 global maximum in 99.70% of 2000 simulated ftMLDE experiments.[61] This represents an 81.1-fold improvement over traditional DE (which achieved the global maximum 1.23% of the time in simulated experiments) and at least a 12.2-fold improvement over the originally published method (which achieved the global optimum 8.17% of the time with a screening burden of 570 total variants—90 more than were used in this work).

This chapter describes improvements to the original MLDE method. It also highlights (1) the importance of considering the unique attributes of fitness landscapes when applying ML to protein engineering problems, (2) the importance of informative training set design for building effective ML models in protein engineering, and (3) how tools developed across a

variety of protein engineering domains can be combined into a cohesive, highly efficient engineering pipeline. To improve access to such a pipeline, I introduce the MLDE software package, made available on the Arnold Lab GitHub (https://github.com/fhalab/MLDE). Designed to be accessible to non-ML and non-computational experts, this repository contains Python scripts that allow execution of MLDE and ftMLDE on arbitrary combinatorial fitness landscapes, thus enabling wet-lab application.

2.2 Results for Chapter 2

2.2.1 MLDE Procedure, Simulated MLDE, and Evaluation Metrics

MLDE attempts to learn a function that maps protein sequence to protein fitness for a multi-site simultaneous saturation mutagenesis ("combinatorial") library (Figure 2-1C). More concretely, MLDE attempts to regress a function f(x) = y describing the fitness landscape of the combinatorial library where the protein sequence is "x" and the protein fitness (i.e., the sequence's label) is "y." I provide detailed information about the programmatic implementation of MLDE in the methods section (A.3: MLDE Programmatic Implementation). Briefly, however, and at a high level, the procedure begins with gathering the sequences and fitnesses of a small subsample from the combinatorial library. These sequence-fitness pairs are then used to train an ensemble of regressors with varied model architecture (roughly, this can be interpreted as fitting a variety of different functions to the fitness landscape) (Methods: *Inbuilt Models*). A variety of models are trained because the shape of the fitness landscape is not known *a priori*; it is thus not possible to confidently recommend which model architectures would be most effective prior to evaluating their effectiveness on the given landscape. The models are evaluated and ranked based on a 5-fold cross-validation error. Predictions from the top-performing trained models in the ensemble (those with the lowest cross-validation error) are then averaged to predict fitness values for the unsampled (unlabeled) variants that were not in the training set. These variants are ranked according to predicted fitness, and the top M are evaluated experimentally to identify the best-performing ones.

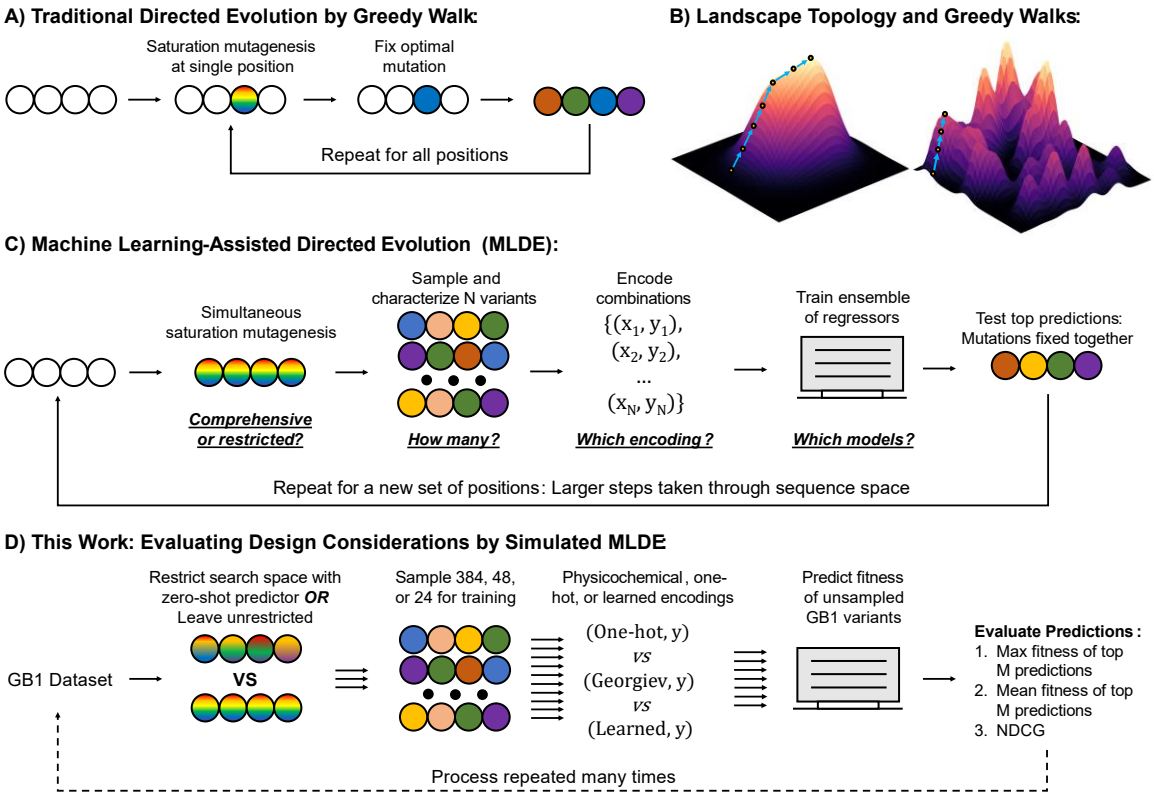


Figure 2-1. Directed evolution strategies and the effects of landscape topology. (A) Directed evolution (DE) by single-mutation greedy walk ("Traditional DE"). In this approach, mutations are fixed iteratively by walking up the steepest fitness gradient. (B) Smooth (left) vs rugged (right) fitness landscapes. A smooth fitness landscape contains a single fitness maximum, so traditional DE is guaranteed to eventually reach the global optimum, though the number of steps needed will depend on the topology of the peak. A rugged fitness landscape contains multiple fitness maxima. Traditional DE is only guaranteed to reach a local fitness optimum here; the maximum achieved will depend on the starting protein variant and the order in which positions are chosen for mutagenesis and testing. (C) Machine learning-assisted directed evolution (MLDE). In this approach, standard molecular biology techniques are used to construct a "combinatorial library" by making mutations at multiple positions simultaneously (e.g., through use of "NNK" degenerate primers). Samples are drawn from this library (e.g., picking colonies from a plate), sequenced, expressed, assayed, and then used to train an ensemble of regressors. This ensemble is used to predict which combinations not seen in the initial draw will have the highest fitness, which are then constructed and tested experimentally. Because the best mutations are fixed simultaneously, MLDE operates in a path-independent manner, so the global optimum of a combinatorial space can be achieved regardless of the starting point. Once mutations are fixed for a given set of positions, a new set is chosen and the procedure is repeated, allowing for larger, more efficient steps through sequence space. The MLDE procedure has many design considerations, which are highlighted as questions under each step. (D) The simulation procedure used throughout this study to evaluate improvements to the MLDE workflow, with the tests performed to evaluate the different design considerations given above each step.

The simulation procedure is repeated many times using data from the GB1 landscape. The effectiveness of a simulated MLDE experiment is determined by (1) evaluating the max and mean true fitness of the top M variants according to predicted fitness and (2) calculating the normalized discounted cumulative gain (NDCG) over all predictions in the simulation.

Throughout this chapter, I evaluate design considerations of MLDE through simulation on the empirically determined four-site combinatorial fitness landscape of protein G domain B1 (GB1). Originally reported by Wu et al., this landscape consists of 149,361 experimentally determined fitness measurements for 160,000 possible variants, where fitness is defined by both the ability of the protein to fold and the ability of the protein to bind human IgG-Fc.[123] To my knowledge, this landscape is the only published one of its kind (i.e., the only almost-complete combinatorial landscape where fitness is reported as scalar values amenable to training the regression models used in MLDE). By imputing the fitnesses of the remaining 10,639 variants and evaluating the resultant complete landscape, Wu et al. identified 30 local optima, the routes to which were often indirect (e.g., if a local optimum was four mutations away from a starting point, it would take more than four mutations to travel by single-mutation greedy walk from the starting point to the optimum). Epistatic interactions are thus highly prevalent in the GB1 landscape. The goal of simulated MLDE is to mimic what would be observed if thousands of MLDE experiments were performed on GB1. Thus, to ensure that the simulations match what would have been observed experimentally had my simulated experiments actually been performed, I do not use the variants with imputed fitness in this study.

A simulated MLDE experiment begins with generating training data (Figure 2-1D). Here, a small set of variants is drawn from the GB1 landscape (values of 24, 48, or 384 are used throughout this study) and their known fitness values are attached (the variants are labeled). This stage of the simulation is analogous to building a combinatorial library (e.g., by using "NNK" degenerate primers to make mutations at multiple positions simultaneously), picking colonies from an agar plate, then sequencing, expressing, and assaying the variants harbored by the colonies. The training data are then fed into the MLDE pipeline and the average predictions of an ensemble of the top three models are used to rank the unlabeled variants

not in the training data (148,977 or more total variants, depending on the number of samples in the training data) by predicted fitness. The quality of the returned ordering is evaluated using a combination of metrics, including (1) the max fitness of the M-highest-ranked variants, (2) the mean fitness of the M-highest-ranked variants and (3) the ranking metric "normalized discounted cumulative gain" (NDCG) (A.2.5: Evaluation Metrics).[129] Whenever reported, mean and max fitness achieved are normalized to the highest fitness in the unlabeled dataset and so can typically be interpreted as a fraction of the global maximum in the GB1 dataset.

Each evaluation metric summarizes different information about the outcome of an MLDE simulation. The max and mean fitness of the M-highest-ranked variants (hereafter also referred to as "max fitness achieved" and "mean fitness achieved") are the most practically relevant in terms of laboratory application of MLDE, as they are analogous to the max and mean fitness that would be observed if the M protein variants predicted to have highest fitness were experimentally evaluated. Consistent realization of high maximum fitness achieved over many simulations indicates that an MLDE design condition is typically effective at finding *at least one* high-fitness variant; consistent realization of a high mean fitness achieved over many simulations indicates that the design condition is typically effective at identifying *many* high-fitness variants. NDCG does not capture specifics about how MLDE can be expected to perform in a laboratory setting, but instead provides a holistic measure of how well a given MLDE design condition is able to identify and rank the most fit variants in the GB1 landscape without the need to set an arbitrary cutoff such as "the top M" predictions.

NDCG is commonly used to assess the quality of information retrieval algorithms such as search engines, a task that parallels the goal of MLDE.[129] To explain, the goal of search engines is to return a list of relatively rare, highly relevant documents identified among a population of many irrelevant ones; the most relevant documents should be provided at the top of the list and the least relevant at the bottom. Because combinatorial fitness landscapes tend to be dominated by zero- and low-fitness variants,[28,78,122] the goal of MLDE is likewise to identify high-fitness (high-relevance) protein variants among a sea of irrelevant ones; the

highest-fitness variants should ideally be the ones ranked highest by MLDE. For both search engines and MLDE, more weight should be placed on correctly identifying and ranking the most-relevant items than the least-relevant as these are the ultimate items of interest. Indeed, NDCG provides just this type of implicit weighting, which is clear from the equation used to calculate it. The equation for NDCG is

$$NDCG = \left(\sum_{i=1}^{N} \frac{f_i}{\log_2(i+1)}\right) \Bigg/ \left(\sum_{i=1}^{N} \frac{f_i'}{\log_2(i+1)}\right), \qquad \text{Eq. 2-1}$$

where the numerator gives the sum of the true variant fitnesses (f) divided by a logarithmic "discount" based on their *predicted* ranking and the denominator gives the sum of true variant fitnesses divided by a logarithmic discount based on a *perfect* ranking. A higher value of NDCG is thus better, and the maximum NDCG possible is "1." Variants with low fitness contribute minimally to the denominator (both due to having a low "f" and, in a perfect ordering, a high logarithmic discount), and so unless they are incorrectly ranked as the very top variants, they will have minimal effect on the score. Correct ranking among high-fitness variants is thus weighted more strongly than correct ranking among low-fitness variants, but incorrect identification of a low-fitness variant as a high-fitness variant is punished. NDCG as an MLDE evaluation metric thus provides a more holistic view of how well models are able to (1) identify the most fit variants and (2) correctly rank those variants.

2.2.2 More Informative Encodings Can Improve MLDE Outcome

Protein sequences must be numerically encoded by a set of features (numerical descriptors that describe a protein sequence) to be used in ML algorithms. The previous implementation of MLDE used one-hot encoding, an uninformative categorical encoding strategy that captures no information about the biochemical relatedness of different amino acids. The descriptiveness of the features used for encoding can affect the outcome of learning, however, by passing in relevant information to an ML model about the similarities and differences between different datapoints. To investigate the effects of more informative

encodings on MLDE, I tested encoding using physicochemical parameters as well as learned protein embeddings.

Physicochemical parameters are manually engineered features that describe amino acid qualities such as hydrophobicity, volume, mutability, etc. Encoding a protein sequence using these features provides an ML model with information on the physicochemical similarities and differences between amino acids. For instance, valine and alanine would have a more similar "hydrophobicity" score than valine and glutamate. In this work, I used the set of physicochemical parameters developed by Georgiev, which is a low-dimensional representation of over 500 amino acid indices from the AAIndex database.[35,36,124]

Unlike manually crafted physicochemical parameters, learned protein embeddings are featurizations of protein sequences that have been *automatically* learned by machine learning models through a strategy known as "representation learning."[22,37,46,110,130] All extant protein sequences have been selected by natural evolution to perform a function that is useful for their host organism. The goal of representation learning is to directly learn features that describe these proteins, thereby capturing a numerical encoding (an embedding) of the essence of what defines a functional and useful protein. Exactly how this learning is accomplished varies, though most strategies and models currently used are adapted from the field of natural language processing and rely on ever-growing protein sequence databases as a source of training data.[22,37,39,45,110] As with physicochemical parameters, learned protein embeddings capture the similarities and differences between specific amino acids; they also, however, capture contextual information about amino acid positions in a protein, with the exact embedding for a given amino acid changing based on the identities of other amino acids in the same protein sequence.[48,55]

A number of studies have been performed to train models for the production of learned protein embeddings.[37] Given a protein sequence, such models output a matrix of values that are then used to encode the protein. In this work, I tested the effectiveness of learned protein embeddings generated from a variety of models of different sizes and architectures made available in the TAPE, ESM, and ProtTrans GitHub repositories.[51,55,58,61] The models tested

from TAPE were trained using 30 million protein sequences from the Pfam database[39] and have varied architectures, including a transformer architecture ("TAPE-Transformer"),[128,131] three separate LSTM-based architectures ("LSTM," "UniRep," and "Bepler"),[52,60,132] and a dilated residual network architecture ("ResNet");[133] all models in TAPE are defined by around 38 million learnable parameters.

Larger models than those in TAPE trained on more sequences can potentially learn a richer representation of protein sequences.[55,134] To test this potential effect of model and training set size on the quality of learned embeddings for MLDE, I also investigated embeddings generated from the state-of-the-art models "esm1b_t33_650M_UR50S" (hereafter referred to as "ESM1b") from the ESM repository as well as "ProtBert-BFD" from the ProtTrans repository. Both of these models have a transformer architecture. ESM1b is a 650-million-parameter model trained on 27.1 million sequences from the UniRef50 database.[42,55] ProtBert-BFD is a 420-million-parameter model trained on 2.1 billion protein sequences from the Big Fat Database (BFD).[51,135] A final model investigated for learned embedding generation was the MSA Transformer, also made available in the ESM repository.[61] Unlike the other models, which were all trained on protein sequences, the 100-million-parameter MSA Transformer was trained on 26 million protein multiple sequence alignments (MSAs); the learned embeddings from the MSA Transformer are thus generated from an MSA of the target protein rather than the target protein sequence alone. MSAs more directly represent information relevant to protein engineering: specifically, related sequences aligned to a reference provide evidence for what mutations are and are not allowed at given positions. I included the MSA Transformer to test if the additional information provided by embeddings generated from an MSA could lead to an improved MLDE outcome.

For each encoding considered, I performed 2000 MLDE simulations at three different training set sizes. The training data, cross-validation indices (i.e., the different folds used for measuring a cross-validation error), and random seeds (values that enable reproducible random number generation) were kept the same for each encoding strategy in each simulation. For a given simulation, the training data consisted of either 384, 48, or 24 GB1

variants drawn at random from the *comprehensive* (consisting of all 149,361 possible GB1 variants) landscape—only the choice of encoding and training set size were considered as design considerations in these experiments. If 384 variants were used for training, the top 96 predictions were tested; if 48 variants were used for training, the top 32 predictions were tested; if 24 variants were used for training, the top 56 predictions were tested (A.2.4.1: Encoding Comparison Simulations). Training using 384 samples and testing 96 predictions evaluates simulations on a scale that approximates the typical experimental screening burdens for standard DE approaches.[82] Because the ultimate goal of using ML in protein engineering is to reduce or eliminate the number of protein variants that must be experimentally characterized, the ability to train an ML model using limited data is also valuable.[64] Training using 24 or 48 samples evaluates the effectiveness of each encoding in this "low-N" setting. I tested 56 or 32 predictions, respectively, to match the total screening burden (80 variants) of an idealized traditional DE pipeline over a four-site landscape where all 20 amino acids at each position are deterministically evaluated. I note that, due to the cost of synthesizing variants individually, deterministic evaluation of mutations is rarely performed, and researchers instead opt to stochastically sample from pools of mutants (thus raising the required screening burden above 80). The total screening burden of deterministic traditional DE does provide, however, a reasonable "low-N" threshold for MLDE.

Violin plots showing the results of the simulated MLDE experiments are provided in Figure 2-2; summary statistics are provided in Data S1 and a pairwise comparison of encoding effectiveness over all simulations is provided in Data S2; I also provide additional figures on the GitHub repository associated with this work that plot the pairwise encoding comparisons. In all, these results show that using a more informative encoding than one-hot can result in an improved MLDE outcome, but not always and depending on the metric and screening burden used to measure MLDE effectiveness. The only two encoding strategies to consistently show at least marginal improvement over the one-hot baseline regardless of metric and screening burden were physicochemical (Georgiev) parameters and learned embeddings from the MSA Transformer. At a training size of 384, NDCG was the only evaluation metric to consistently suggest that more informative encodings improve MLDE

outcome. For the max fitness achieved, simulations run using Georgiev parameters and learned embeddings from the MSA Transformer tended to achieve marginally higher max fitness than those run using one-hot encodings; simulations run using all other learned embeddings tended to achieve the same if not a slightly lower one. For the mean fitness achieved, simulations run using the embeddings from the Bepler model as an encoding strategy slightly underperformed one-hot, those using embeddings from UniRep and ProtBert-BFD performed comparably, and those using all other encoding strategies tended to achieve a higher mean fitness than one-hot.

For smaller training set sizes, the effect of different encodings on MLDE outcome was less noticeable. Only simulations run using embeddings from the TAPE transformer and the MSA Transformer still obtained a higher NDCG than those run using one-hot encoding; simulations using other encodings tended to yield comparable to marginally better NDCG than those run with one-hot. For the max fitness achieved, all non-MSA Transformer learned embeddings arguably gained ground on one-hot and Georgiev encodings, though the results are still comparable at best. The opposite was observed for the mean fitness achieved, with one-hot typically gaining ground on and even slightly surpassing many learned embeddings.

My results largely agree with recent work suggesting that training ML models using existing learned protein embeddings yields marginal improvement at best compared to using simpler encodings such as one-hot or physicochemical parameters.[62,63] Unlike previous works, however, which found that learned embeddings tended to be superior to simpler strategies in the low-N regime,[62,64] I found that simpler strategies remained competitive regardless of training set size. It is possible that taking an "evotuning" strategy like that of Biswas et al., where embedding models are further trained on sequences more closely related to the target protein (i.e., GB1), could improve the performance of learned embeddings in the low-N regime;[64] however, as will be discussed in greater detail in later sections, this possibility is currently untestable due to the limited availability of GB1 homologs in existing sequence databases.

I also found minimal benefit in using embeddings derived from the larger models trained on larger corpora of protein sequences. For instance, when trained with 384 samples, simulations run using embeddings from ESM1b only slightly outperformed those run using embeddings from the TAPE transformer, despite the ESM1b model being ~17-fold larger; in the low-N regime, simulations run using embeddings from ESM1b underperformed those run using encodings from the TAPE transformer. Likewise, regardless of training set size, simulations run using embeddings from ProtBert-BFD often underperformed many of the TAPE models, despite ProtBert-BFD being ~11-fold larger and trained using ~70-fold more protein sequences. Indeed, the most effective model for generating learned embeddings was the MSA Transformer, which is 1/6 the size of ESM1b and 1/4 the size of ProtBert-BFD. As mentioned above, the MSA Transformer was trained on—and generates embeddings using—MSAs rather than protein sequences. It is possible that the additional information provided by the MSA yields more effective learned embeddings for MLDE, though this is impossible to conclude working off of just the GB1 dataset. It does stand to reason, however, that models and data sources that more directly represent information known to be important for protein function could lead to embeddings that are more informative for MLDE. Developing such data sources and models is a potentially valuable avenue for future research.

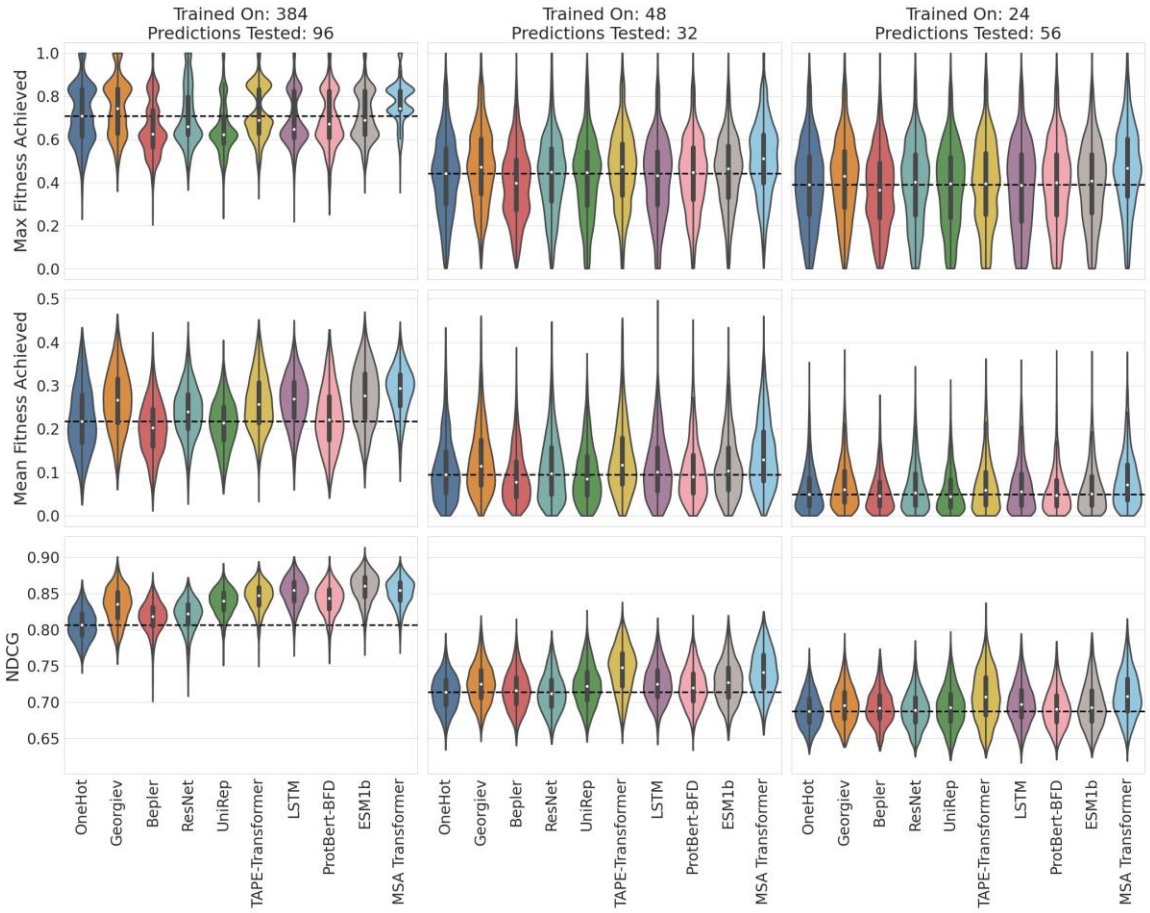


Figure 2-2. More informative encodings can improve MLDE outcome: results of simulated MLDE comparing ten different encoding strategies at three different screening burdens. Note that, for the sake of computational efficiency, 19 of the 22 inbuilt MLDE models were in the ensemble trained for simulations using the large TAPE transformer-, the MSA Transformer-, ESM1b-, ProtBert-BFD-, UniRep-, and LSTM-derived encodings, while 22 were in the ensemble for all others. Each column of plots gives the results of a different screening burden. Each row of plots gives the results for a different summary metric. Rows and columns share the same axes. Each violin represents the results of 2000 simulated MLDE experiments, and the dashed line represents the median summary value of simulations run using one-hot encoding. In general, whether or not an encoding strategy outperformed the one-hot baseline depended on the screening burden tested and summary metric evaluated. The only two encoding strategies to consistently show at least marginal improvement over the one-hot baseline regardless of metric and screening burden were physicochemical (Georgiev) parameters and learned embeddings from the MSA Transformer. For summary statistics of simulation results, see also Data S1. For pairwise comparisons of simulation results, see also Table A-1 – Table A-4, Data S2, and additional figures at the GitHub associated with this work.

2.2.3 Models/Training Procedures More Tailored for Combinatorial Fitness Landscapes Can Improve MLDE Predictive Performance

Many of the learned embeddings used in the previous section are extremely high dimensional, with the largest (LSTM) describing each combination of four amino acids with 8192 features (Table A-1). To better handle the high dimensionality introduced by learned embeddings, in this new implementation of MLDE I added two 1D convolutional neural network (CNN) architectures to the ensemble of models trained, one with a single convolutional layer and another with two (A.3.1: Inbuilt Models). CNNs apply sliding windows ("convolutions") over structured, high-dimensional data, relying on spatial dependencies between elements of the input data to extract the most relevant high-level features.[136,137] For instance, CNNs are often applied to image processing tasks, where sliding 2D windows are used to extract high-level features by aggregating information from local groupings of pixels. CNNs can also be applied, however, to sequential data such as protein and DNA sequences. When applied to proteins, the sliding windows are 1D (hence, "1D CNN") rather than 2D, and are applied over the protein sequence to extract high-level features by aggregating information from nearby members of the sequence.[19,125,138] The practice of using a sliding window to extract or aggregate information from sequences or sequence alignments has been used in bioinformatic analyses for decades.[139–141] Whereas sliding windows have historically been used to extract specific, human-defined information, the sliding windows of 1D CNNs automatically learn the aggregate information most relevant for relating a sequence to a label (e.g., the high-level features that relate protein sequence to fitness). Recent evidence suggests that 1D CNNs are a particularly effective model class for protein engineering.[19] I found that 1D CNNs could be beneficial for MLDE, but that the specific architecture of the 1D CNN and training points used to train it were important. For instance, when trained with 384 training points, the two-layer 1D CNN was consistently among the top-ranking models in terms of cross-validation error during training, particularly for higher-dimensional encodings (Table A-1). The same could not be said, however, for the single-layer 1D CNN or the two-layer 1D CNN trained with less data (Table A-1 – Table A-2).

In addition to 1D CNN architectures, I also integrated XGBoost models trained with the Tweedie regression objective to better handle the zero-inflated nature of fitness landscapes.[126,142,143] XGBoost is a Python package that implements the gradient boosting technique, which, at a high level, is a strategy of combining multiple weak predictors (multiple weak models) to create a more effective predictor.[126] Gradient-boosted Tweedie regression was developed to handle regression for datasets with zero-inflated labels.[142,143] Because most mutations are deleterious to activity or stability, as more mutations are made to a protein, the probability that it will still fold and function drops.[122] The result is that combinatorial fitness landscapes tend to be dominated by proteins with zero or extremely low fitness,[28,78] something that is highlighted by the distribution of fitnesses for GB1 (Figure 2-3A). Training data drawn from combinatorial fitness landscapes will thus also have an over-abundance of zeros, which can bias ML models to be more effective at predicting low-fitness variants than high-fitness ones. To test if implementing the Tweedie regression objective could improve the effectiveness of XGBoost models in MLDE, I included XGBoost models trained with both the Tweedie and default (root mean squared) training objectives in the ensemble of models trained in the simulations discussed in the previous section. I found that models trained with the Tweedie objective on average achieved a higher NDCG than models trained with the default objective regardless of base model (the architecture of the weak predictors used by XGBoost) and encoding; only models with a tree base model on average showed improved max and mean fitness achieved, however (Table A-3 – Table A-4). Additional supplemental images plotting a pairwise comparison of the results of XGBoost simulations run with each learning objective can also be found at the GitHub repository associated with this work.

2.2.4 The Challenge of Holes in Combinatorial Fitness Landscapes and the Importance of Informative Training Data

Diversity within training data is critical to constructing an effective machine learning model. Often, training set diversity is thought of in terms of exploration of the feature space, where limited resources are intelligently committed to minimize the amount of extrapolation that must be performed when making predictions (Figure 2-3B). For instance, for protein

engineering, a researcher would aim to experimentally characterize diverse protein sequences when gathering data to train an ML model; a model trained on a restricted set of sequences may struggle to generalize to more diverse sequences when used for prediction. Equally important to feature diversity, however, is diversity in the labels: patterns in the ground truth will not be identified if there are no patterns in the training data (Figure 2-3B). The overabundance of "dead" (zero- or very low-fitness) variants in combinatorial fitness landscapes thus poses an additional challenge beyond that discussed in the previous section: a random draw for the generation of training data is likely to be populated by primarily zero- or extremely low-fitness variants. And, while potentially useful for classifying dead vs functional proteins, these "holes" provide no information about the *extent* to which specific combinations of mutations benefit or harm fitness—only that fitness is destroyed by a combination—and so have limited utility when training the regression models used in MLDE.

I thus proposed a general strategy of running MLDE with training sets designed to contain a minimal number of holes. In this strategy, which I call "focused training MLDE" (ftMLDE), training data is not randomly drawn from the full combinatorial landscape (which will return primarily holes), but is instead drawn from diverse regions of sequence space believed to contain functional variants. A training set drawn in this way will consist of a greater proportion of functional variants and so will provide more information to an ML model about the magnitude of the effects of different mutations on fitness, enabling more effective regression of a function to the fitness landscape.

To demonstrate the concept of ftMLDE and test its effectiveness, I designed training data enriched in functional, but not the fittest, protein variants, and then used it to perform simulated ftMLDE. Because I had access to the full GB1 dataset, I could choose what data to use for training. As such, I built training sets consisting of 384 samples where 50% of the variants had fitness greater than or equal to a given threshold of either 0.011, 0.034, 0.057, or 0.080 and 50% had fitness below (A.2.4.2: High-Fitness Simulations). A higher fitness threshold thus meant greater fitness enrichment in the training data (greater "focus" of the

training data on higher-fitness regions of the protein fitness landscape) and vice versa (Figure A-1). To avoid "cheating" by inclusion of the highest-fitness variants in the training data, I also enforced a requirement that no variant in the training data had fitness greater than 34% of the global maximum. By including this upper limit, the highest-fitness variants in the GB1 landscape could only be identified from model predictions.

The results of 2000 simulated ftMLDE experiments using training sets from each of the four considered thresholds are given in Figure 2-3C-E and Table A-5; also included are the results of 2000 simulated standard MLDE experiments where training data were randomly drawn from all variants with fitness below 34%. Compared to standard MLDE, the ftMLDE simulations show improved NDCG, mean fitness achieved in the top 96 predictions, and max fitness achieved in the top 96 predictions. Training data enrichment using even the lowest fitness threshold (0.011) led to improvement in evaluation metrics, with NDCG increasing ~8%, the max fitness achieved improving ~19%, and the mean fitness achieved improving ~49%. The lowest threshold sits at just above 1% of the fitness of the global maximum, and so the improvement observed here suggests that even the weakest degree of enrichment can lead to an improvement in engineering outcome. Indeed, while further increasing the fitness threshold did further improve outcome, the degree of improvement was not as large. For instance, increasing the threshold from 0.011 to 0.080 led to a further ~2% increase in NDCG, ~4% increase in max fitness achieved, and ~10% increase in mean fitness achieved, roughly 5-fold less overall improvement compared to moving from no threshold to a threshold of 0.011. This result suggests that, while achieving a higher mean fitness in the training data is beneficial to ftMLDE, the more important factor is elimination of holes.

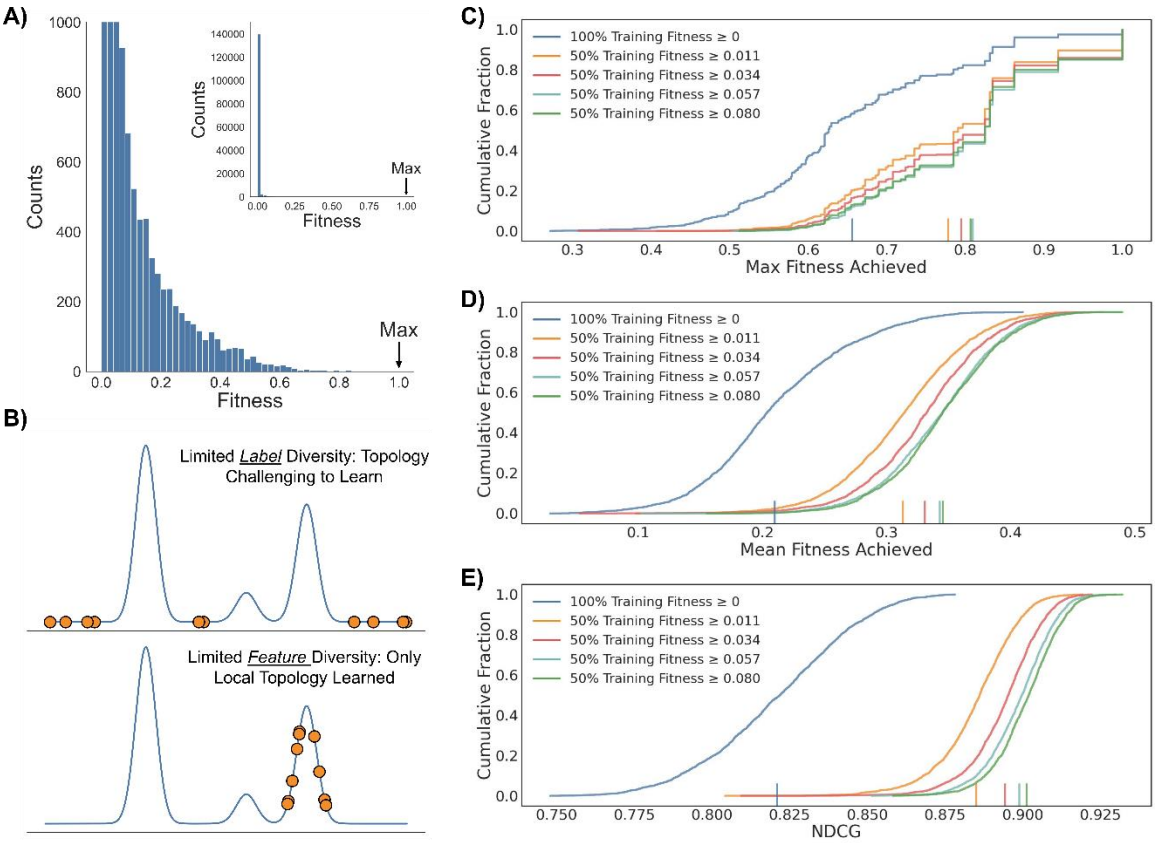


Figure 2-3. The challenge of holes in combinatorial fitness landscapes and the importance of informative training data. (A) The distribution of fitness in the GB1 landscape shown as a histogram. Most variants in this epistatic landscape have extremely low fitness, and the highest-fitness variants are very rare. (B) A demonstration of the importance of diversity in both the labels and features of training data for machine learning. Learning detailed topology is challenging if the labels are not representative of it, even if sampled from diverse regions of feature space. Only local topology can reliably be learned if points are sampled from a restricted region of feature space. (C) The maximum fitness achieved for simulated ftMLDE using training data designed to be enriched in fit protein variants. Specifically, training sets were designed such that 50% of the variants had fitness greater than or equal to a given threshold of either 0.011, 0.034, 0.057, or 0.080 and 50% had fitness below. A higher fitness threshold enforces a higher mean training fitness. All data is shown as empirical cumulative distribution functions (ECDFs); vertical lines on the x-axis give the expectation value of the distribution. Each ECDF represents the results of 2000 simulated ftMLDE experiments. (D) The mean fitness achieved for simulated ftMLDE using training data enriched in fit variants. (E) The NDCG for simulated ftMLDE using training data enriched in fit variants. See also Figure A-1 and Table A-5.

2.2.5 Zero-Shot Prediction as a Practical Training Set Design Strategy for ftMLDE

Of course, in practice, the full dataset for a combinatorial library would not be available as it is for GB1, otherwise there would be no point in applying MLDE in the first place. Instead, the protein variants used to build ftMLDE training data must be chosen *prior* to knowing their fitnesses—practical application of ftMLDE requires at least a weak predictor of protein fitness for training set design. One way to accomplish this would be to take an active learning approach. That is, using data from a prior round of standard MLDE performed for the same combinatorial library, a model could be trained to predict a diverse set of higher-fitness variants; these variants could then be experimentally evaluated and used to train models in a round of ftMLDE. Indeed, a strategy like this was taken by Romero et al. when evolving for improved P450 thermostability, where a classifier trained on data from one round of evolution was used to build a training dataset enriched in functional protein variants for the next round of evolution.[78] While this active learning approach has proven successful, it adds an additional round of data collection to the workflow, which is undesirable. I thus chose to investigate zero-shot prediction strategies for training set design.

I define zero-shot prediction strategies as those capable of predicting protein fitness without the need for further labeled training data collection, and thus they do not affect the overall screening burden of ftMLDE. A number of zero-shot strategies exist for protein functional prediction, ranging from scoring protein variants based on evolutionary sequence conservation,[68,70,144] to generative modeling,[57,68,145] and physics-based computational modeling (e.g., prediction of $\Delta\Delta G$ upon mutation),[146–150] to name a few. The remainder of this chapter is devoted to evaluating the effectiveness of different zero-shot strategies for designing training data for ftMLDE. Over the next two sections, I evaluate zero-shot strategies from each of the aforementioned overarching zero-shot classes for their ability to predict GB1 fitness. In the final section, I use the successful zero-shot predictors to demonstrate a practical application of ftMLDE to the GB1 landscape. I find that ftMLDE is superior to both standard MLDE and traditional DE, with the best ftMLDE condition achieving the global maximum in 99.70% of simulated experiments compared to 8.85% for the best standard MLDE condition and 1.23% for simulated traditional DE.

2.2.6 Leveraging Sequence Data for the Design of Fitness-Enriched Training Data

Over billions of years, natural evolution has tested countless protein sequences, discarding those that were detrimental to a host organism and propagating those that were beneficial. The list of extant protein sequences represents those that survived the filters of evolution and so implicitly contains information about the evolutionary and biophysical rules that enable production of a useful protein. Driven by a combination of increased computational power and greater availability of sequence data, recent years have seen renewed effort to extract this implicit fitness information contained in sequences and use it to reduce or eliminate the amount of experimentally acquired sequence-fitness data needed for reliable prediction of protein fitness. All of these strategies assume that a given list of functional protein sequences is representative of a distribution of allowed protein sequences and that by learning this distribution the fitness of a new protein sequence can be inferred. Specifically, a new sequence highly likely to belong to the learned distribution is predicted to have high fitness and vice versa.[22,68,70] The simplest example of a sequence-based zero-shot strategy, for instance, is use of BLOSUM matrices, which score the likelihood of a given amino-acid substitution based on observed substitution frequencies in conserved protein families.[151] Far richer strategies than BLOSUM matrices have been developed, however, and in this section, I test the ability of a number of them for zero-shot prediction of GB1 fitness.

Strategies for sequence-based zero-shot prediction can be broadly classified as relying on local or global sequence information. Local strategies attempt to learn the distribution of allowed sequences from those related to a target. These strategies first search sequence databases to build an MSA against the target, then use that MSA to learn a representation of the underlying sequence distribution defining allowed local protein sequences. Global strategies, in contrast, attempt to learn the distribution of allowed sequences from large databases of unrelated protein sequences. The language models trained to build embeddings of protein sequences are examples of global strategies. Indeed, a proposed and assumed rationale for the benefit of embeddings derived from natural language processing models is that the embedding vectors learned during training should capture the global rules of what defines a functional protein.

I first tested the local sequence-based zero-shot predictors EVmutation and DeepSequence.[68,70] Among other requirements, the authors of these tools recommend training using an MSA with ≥10L (where "L" is the length of the target protein) redundancy-reduced sequences (essentially, a measure of the effective number of sequences given the diversity of those in the MSA—less diverse MSAs have a lower number of redundancy-reduced sequences) that cover the positions at which the effects of mutations are to be predicted; at least 560 redundancy-reduced sequences are thus the target for 56-amino acid-long GB1. There are, unfortunately, few recorded sequences that are homologous to GB1, and I could at best produce an MSA with 56 redundancy-reduced sequences that covered all four positions of interest in the GB1 combinatorial landscape (A.2.1: Alignment Generation and EVmutation Model Training). Despite this relatively uninformative MSA, however, EVmutation still performed reasonably well as a zero-shot predictor, achieving a Spearman rank correlation coefficient (Spearman ρ) of 0.21 (Figure 2-4, Table A-6). DeepSequence, in contrast, was less effective, achieving Spearman ρ = 0.05 (Table A-6, A.2.3.1: EVmutation/DeepSequence Calculations). These results align with an observation in the original DeepSequence publication, where DeepSequence was shown to be more susceptible to failure than EVmutation when trained on low-quality MSAs.[68] This is not to say that the predictions of EVmutation were unaffected by the low-diversity GB1 MSA. The low information content of the MSA made it impossible for EVmutation to assign unique probabilities of fitness to all GB1 combinations, resulting in the coarse ranking pattern shown in Figure 2-4.

For global sequence-based zero-shot predictors, I tested a mask filling protocol for each of the models made available in the ESM GitHub repository as well as the ProtBert and ProtBert-BFD models from the ProtTrans GitHub repository.[55,56,61] All of these models were trained using a protocol known as "masked token prediction."[128] When training using this protocol, a model is fed a sequence with the identities of amino acids at a fraction of its positions obscured ("masked"). Given the context of the unobscured ("unmasked") amino acids, the objective of the model is to then predict the correct original identities of the masked amino acids by modeling the probability $P(s_{masked}|s_{unmasked})$, where s is a sequence of

amino acids. By repeating this procedure over millions (or billions, in the case of ProtBert-BFD) of sequences, the model learns a global sense of the distribution of allowed proteins: in particular, it learns the probability that a given combination of amino acids will occur in the context of a given sequence background. Using a trained model, masked token prediction can be co-opted for zero-shot prediction using a "mask filling protocol." Specifically, given a sequence with positions of interest masked, the model can be used to predict $P(s_{masked}|s_{unmasked})$ for all possible combinations of mutations at the masked positions. Combinations of mutations with higher probability are then assumed to have higher fitness as they more accurately represent the learned distribution of allowed amino acid combinations.

The models in the ESM repository, in combination with ProtBert and ProtBert-BFD, are a variety of different sizes and were trained on varying numbers of sequences, allowing me to test the effect of model capacity on the mask filling protocol for GB1. Additionally, the ESM repository contains the MSA Transformer which, uniquely, was trained using MSAs produced for each sequence in the UniRef50 database, making it somewhat of a hybrid between a local and global sequence model.[42,61] I included the MSA Transformer in the mask filling zero-shot predictions to see if the global information captured during training could make up for the limited information provided by the small GB1 MSA used for EVmutation and DeepSequence predictions.

For zero-shot prediction with a mask filling protocol, I calculated $P(s_{masked}|s_{unmasked})$ for every combination in the GB1 landscape using either naïve or conditional probability (A.2.3.2: Mask Filling Protocol). Note that, for all non-MSA Transformer methods, the parent GB1 sequence was used to define $s_{unmasked}$, while for the MSA Transformer, the MSA used for EVmutation (with slight additional processing, see Section A.2.3.2 for details) was used to define $s_{unmasked}$. The MSA Transformer thus had access to additional local evolutionary information when making mask-filling predictions. The results using both naïve and conditional probability protocols for all tested models are provided in Table A-7; the results using naïve probability with the MSA Transformer are also depicted in Figure 2-4. In

all cases, I found that predictions made using naïve probability to be slightly superior to predictions made using conditional probability. The naïve probability prediction procedure more closely mimics the masked token prediction procedure used to train the ESM and ProtBert models, providing a potential explanation for its slight superiority over the conditional prediction procedure, though a conclusive reason for this observation is not immediately clear.

Differences in effectiveness between the naïve and conditional probability predictions notwithstanding, for all models except the MSA Transformer, mask filling was an ineffective zero-shot prediction strategy. Indeed, the predictions from most models gave a negative correlation (Spearman ρ) with GB1 fitness, indicating a prediction that is worse than a random guess. Additionally, and perhaps contrary to expectations, smaller models trained on the same data with the same training procedure tended to outperform larger ones. Specifically, predictions using esm1_t6_43M_UR50S (43 million parameters) outperformed those using esm1_t12_85M_UR50S (85 million parameters) which in turn outcompeted those using esm1_t34_670M_UR50S (670 million parameters). Correlations between the amount of training data and zero-shot prediction performance are less apparent. For instance, even though zero-shot predictions using esm1_t34_670M_UR100 (trained on UniRef100) outcompeted those using esm1_t34_670M_UR50 (trained on UniRef50, and otherwise equivalent to esm1_t34_670M_UR100), predictions using ProtBert-BFD (trained on BFD) were more or less as effective as those using ProtBert (trained on UniRef100, and otherwise equivalent to ProtBert-BFD).

The exception to the general failure of mask filling as a zero-shot predictor was those predictions generated by the MSA Transformer (Figure 2-4), which achieved a Spearman ρ of 0.24 with naïve probability (0.20 with conditional). Even though this Spearman ρ is comparable to that achieved using EVmutation, it is notable that the many ties observed in the EVmutation predictions are not present in the mask filling zero-shot predictions from the MSA Transformer, presumably due to the global information captured by the MSA Transformer during training. A mask filling protocol using the MSA Transformer could thus

be an attractive zero-shot alternative to EVmutation for proteins for which deep, high-quality MSAs cannot be produced. It must, of course, also be asked whether the underrepresentation of GB1 homologs in sequence databases leads to the failure of mask filling zero-shot prediction by non-MSA Transformer models. Answering this question is impossible using just the GB1 landscape alone, however, and would require access to other combinatorial landscapes built in proteins with varying degrees of representation in the sequence databases used to train the ESM and ProtBert models.

2.2.7 Predicted ΔΔG of Stabilization for the Design of Fitness-Enriched Training Data

Just because a sequence motif is not represented in a sequence database does not necessarily mean that it would be detrimental to a protein's function. It is possible, for example, that a natural function that would benefit from such a motif does not exist, that evolution has not yet explored such a region of sequence space, or simply that humans have not yet sequenced a representative protein. The underlying assumption of sequence-based zero-shot strategies that evolutionarily optimized fitness correlates to a target fitness may thus not always hold. In such cases, using a zero-shot strategy such as predicted ΔΔG of stabilization upon mutation may be beneficial.[146–150] This approach attempts to calculate the effect of a mutation on protein stability from first principles. Based in physics, it is thus not subject to the assumption that the target fitness correlates with the fitness of existing proteins, but instead that protein stability plays a role in fitness.

The fitness of GB1 is considered to be, at least in part, a function of stability, suggesting that approaches like predicted ΔΔG of protein stability upon mutation might be an effective zero-shot predictor.[123,152] Indeed, I found a correlation between single-mutant fitness data and literature GB1 ΔΔG data (|Spearman ρ| = 0.58, Figure A-2A).[153] Wu et al. also previously presented evidence suggesting that predicted ΔΔG could be correlated to GB1 fitness.[123]

To test the effectiveness of ΔΔG predictions as a zero-shot predictor for GB1 fitness, I used the Triad protein design software suite (Protabit, Pasadena, CA, USA: https://triad.protabit.com/) with a Rosetta energy function to predict the stability of each of the 149,361 GB1 variants with measured fitness, then calculated a predicted ΔΔG of

stabilization for each variant relative to the parent amino acid sequence (A.2.3.3: ΔΔG Calculations). Both fixed backbone and flexible backbone calculations were performed using a previously determined GB1 crystal structure (PDB: 2GI9) as a scaffold.[154] The predicted ΔΔGs from each calculation correlated with literature values of experimentally determined ΔΔG values for the single mutants, though the fixed backbone calculations were more effective (Spearman $\rho = 0.61$ for fixed backbone, Spearman $\rho = 0.42$ for flexible backbone, Figure A-2B–C). Despite both approaches having predictive power for single mutant ΔΔG, only the fixed backbone calculations were effective at identifying GB1 variants enriched in fitness when ranking by predicted ΔΔG (Spearman $\rho = 0.27$, Figure 2-4, Table A-6, Figure A-3). If instead, however, the GB1 variants were ranked by root mean squared deviation (RMSD) of variant structures produced during flexible backbone calculations, those variants with the lowest RMSD tended to be enriched in fitness, though not as strongly as in the fixed backbone calculations (Spearman $\rho = 0.06$, Table A-6, Figure A-4).

Structurally conservative mutations are generally less likely to disrupt protein function, and so the observation that RMSD can be used for zero-shot prediction is not entirely surprising. Because fixed backbone calculations will tend to heavily penalize mutations that would require large backbone movements to stabilize, an interesting question arises over the extent to which structural conservation or accurate prediction of ΔΔG allows effective fixed backbone zero-shot prediction of fitness in GB1. If structural conservation dominates, it is possible that Triad could be used for zero-shot prediction with other combinatorial libraries in proteins where variant fitness is not related to stability, particularly when the mutated residues in question are tightly packed together and/or buried in the protein core as they are for the GB1 landscape used in this study (Figure A-5). Answering this question and evaluating the generalizability of fixed backbone Triad calculations is beyond the scope of this work, but as more fully combinatorial datasets become available this question should be investigated further.

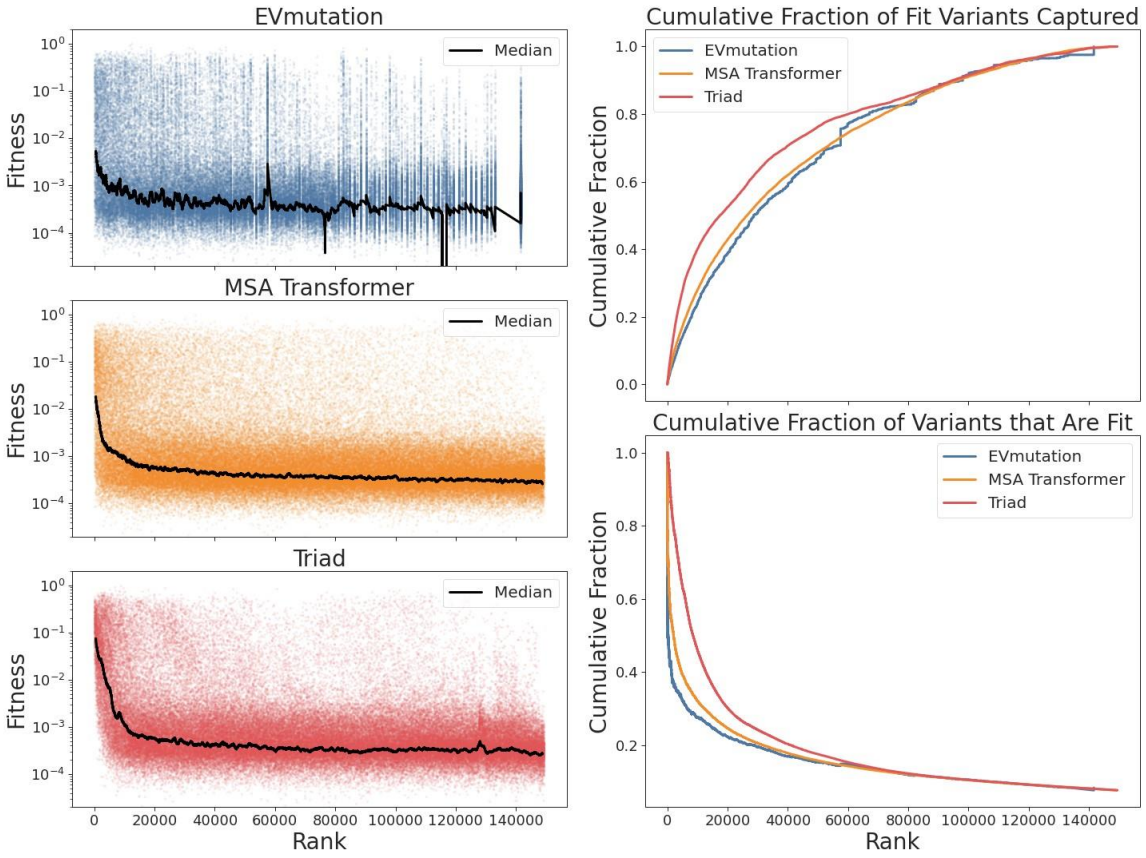


Figure 2-4. Zero-shot prediction for the design of fitness-enriched training data. All figures plot the predicted rank of GB1 variants (where the variants predicted to be most fit have lower rank and vice versa) against either fitness (A–C) or an alternate summary metric (D–E). In A–C, dots are all individual variants while the black line is the sliding median (window size = 1000) of fitness. (A) Results of zero-shot prediction using EVmutation. (B) Results of zero-shot prediction using a mask filling protocol with the MSA Transformer. (C) Results of zero-shot prediction using predicted ΔΔG from Triad with a fixed protein backbone. (D) The fraction of all fit variants in the GB1 landscape captured up to and including a given rank. A "fit" variant is defined as one with fitness greater than 0.011 (which was the lowest threshold tested for the simulations performed with training data designed to be higher in fitness—see Figure 2-3). (E) The cumulative fraction of fit variants captured up to and including a given rank. See also Table A-6 and Table A-7 and Figure A-2 – Figure A-5.

2.2.8 Zero-Shot Predictions for Training Set Design Enable Highly Effective ftMLDE on the GB1 Landscape

As a final demonstration, I evaluated the performance of ftMLDE using GB1 training data predicted to be higher in fitness by the three successful zero-shot prediction strategies: EVmutation, mask filling using the MSA Transformer, and Triad ΔΔG calculations. To begin, I generated training data by randomly sampling 2000 training sets of 24, 48, and 384 variants from the top 1600 (1.1%), 3200 (2.1%), 6400 (4.3%), 9600 (6.4%), 12,800 (8.6%), 16,000 (10.7%), and 32,000 (21.4%) variants as ranked by each zero-shot predictor; completely random training data (i.e., from the full landscape) were also drawn at each sample size so that standard MLDE could be performed as a control. These splits resulted in 66 total "training data types" to test (three random MLDE training data types plus 21 zero-shot ftMLDE training data types for each of three zero-shot predictors), each made up of 2000 training sets. Predictive algorithms (zero-shot predictors included) will tend to predict that similar sequences have similar fitness, so sampling from different percentiles of the top predictions explores the exploration-exploitation tradeoff of using zero-shot predictions for training set design. In other words, sampling from a larger top percentile of the ranked variants allows greater sequence diversity in the training data (thus potentially enabling exploration of more fitness peaks as depicted in Figure 2-3B) at the expense of confidence that the variants will have non-zero fitness (Figure A-6). While I previously used training sample sizes of 24 and 48 to test the effectiveness of different encodings in the low-N setting, here I included them to enable comparison of ftMLDE (and standard MLDE) with the most efficient implementation of traditional DE. As discussed previously in the encoding comparison section, traditional DE can in principle be performed on a four-site library by deterministically evaluating all 20 amino acids at each position, requiring only 80 measurements for the GB1 landscape. Again, due to the cost of synthesizing variants individually, this approach is rarely taken. However, use of 24- and 48-variant training sets (with 56 and 32 tested predictions, respectively) allows for direct comparison of the *algorithms* of ftMLDE and this most efficient implementation of traditional DE.

For each of the 66 training data types, simulated MLDE was performed using each training set with variants encoded using either one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer (which was the most effective of the learned embeddings tested earlier) (A.2.4.3 Zero-Shot Simulations). In total, testing all encodings with all training data types amounted to 198 "training conditions" (66 training data types × 3 encodings/type) and 396,000 simulated MLDE experiments (198 training conditions × 2000 simulations/condition = 396,000 simulated MLDE experiments). As before, cross-validation indices and random seeds were kept the same between simulations using different encodings but the same training data. For each simulation, after prediction, only the top-predicted unsampled combinations that could be constructed by recombining combinations in the training data were evaluated (e.g., if "AAAA" and "CCCC" were the only training examples, then only "AAAC," "AACC," "CAAA," etc., could be in the top M proteins chosen for fitness evaluation). This approach enforced a confidence threshold on the predictions and focused all resources on regions believed to contain the highest-fitness protein variants.

The distributions of the achieved max and mean fitnesses for all simulations with a training sample size of 384 are shown in Figure 2-5 and Figure 2-6, respectively. Distributions of the achieved max and mean fitnesses for simulations with smaller training sample sizes of 24 and 48 are shown in Figure A-7 – Figure A-10 and summary statistics for all simulations are provided in Data S3. Both MLDE and ftMLDE using 384 training samples outperformed traditional DE regardless of encoding and zero-shot strategy, with the most effective set of simulations (ftMLDE run using training data sampled from the top-3200 Triad predictions and the MSA Transformer for encoding) achieving the global maximum in 99.70% of simulations. By comparison, simulated traditional DE on the GB1 landscape reached the global optimum just 1.23% of the time (A.2.4.4: Traditional Directed Evolution Simulations). At lower screening burdens, both MLDE and ftMLDE remained competitive with traditional DE (in terms of mean- and median- maximum fitness achieved over all simulations), though only ftMLDE simulations ever achieved the global optimum more frequently than traditional DE. Specifically, ftMLDE simulations using 24 training samples achieved the GB1 global optimum more frequently than traditional DE in 40 out of 63

ftMLDE training conditions; ftMLDE simulations using 48 training samples achieved the GB1 global optimum more frequently than traditional DE in 57 out of 63 training conditions tested. Almost all training conditions where ftMLDE did not outcompete traditional DE in the low-sample setting used mask filling with the MSA Transformer as the zero-shot predictor, with 0 out of 21 such conditions outcompeting traditional DE at a training sample size of 24 and 15 out of 21 at a training sample size of 48. It is also notable that training on 48 samples and testing 32 tended to be a more effective strategy than training on 24 samples and testing 56, as this result indicates that, at least for GB1, devoting screening resources to the training stage of the MLDE workflow may be more important than the testing phase. Indeed, the most effective set of ftMLDE simulations at low screening burden was with 48 training samples (from the top-3200 Triad predictions and encoded using Georgiev parameters), where the global maximum was achieved 9.95% of the time.

Aside from comparisons to traditional DE, the results of the simulations allow direct comparison of ftMLDE and MLDE and show that ftMLDE is generally a more effective strategy for navigating the GB1 landscape than MLDE. The optimal MLDE strategy, for instance, achieved the global optimum just 8.85% of the time compared to the 99.70% of the optimal ftMLDE strategy. Additionally, in almost all training conditions tested, ftMLDE tended to achieve higher mean and max fitness than the comparable MLDE control. There are some exceptions, however, that suggest that the combination of training set diversity, zero-shot strategy, and encoding have an effect on the outcome of ftMLDE. For instance, all ftMLDE training conditions tended to achieve a higher max fitness than the relevant MLDE control except those using 384 training points derived from the top-1600 Triad samples and encoding with one-hot or Georgiev parameters. Similarly, ftMLDE tended to achieve a higher mean fitness than the relevant MLDE control in all training conditions tested except for a number using learned embeddings from the MSA Transformer for encoding with 384 training points derived from sequence-based zero-shot predictors (EVmutation and mask filling using the MSA Transformer).

The reasons for the observed exceptions to ftMLDE's general superiority over MLDE are not immediately clear, though it is interesting to note that (1) the only training condition run using 384 training points from the top-1600 Triad samples that achieved higher max fitness than MLDE was the one using MSA Transformer encodings and (2) the training conditions where ftMLDE achieved a lower mean fitness than MLDE were those trained using data encoded with the MSA Transformer that was derived from sequence-based zero-shot predictors. During training, the embeddings of the MSA Transformer were developed to be able to predict the identity of masked amino acids (See a discussion of the masked-token training procedure above in Section 2.2.6: Leveraging Sequence Data for the Design of Fitness-Enriched Training Data).[61] The embeddings themselves thus contain information about what mutations are and are not likely in a given reference protein based on available sequence data. Indeed, if they did not, I would be unable to successfully make zero-shot predictions using a mask filling protocol with the MSA Transformer model. It is interesting to ask, then, if models trained on data derived from zero-shot predictions made by Triad and encoded using MSA Transformer embeddings have access to two sets of prior information (one derived from the data via physics-based Triad calculations and another from sequence-conservation captured in the MSA Transformer embeddings), thus making them more effective than models trained with encodings that do not capture fitness information from sequence. Similarly, it could be asked if models trained on data derived from sequence-based zero-shot predictors and encoded by the MSA Transformer become overly restricted by a "double-dose" of sequence-based prior information. Such effects could explain both observations at the beginning of this paragraph, and, indeed, why the combination of Triad-derived data and the MSA Transformer was the most effective ftMLDE strategy tested. This is, of course, conjecture, and the only clear conclusion that can be derived from these results is that there is an interplay between training data makeup and encoding strategy in determining ftMLDE outcome.

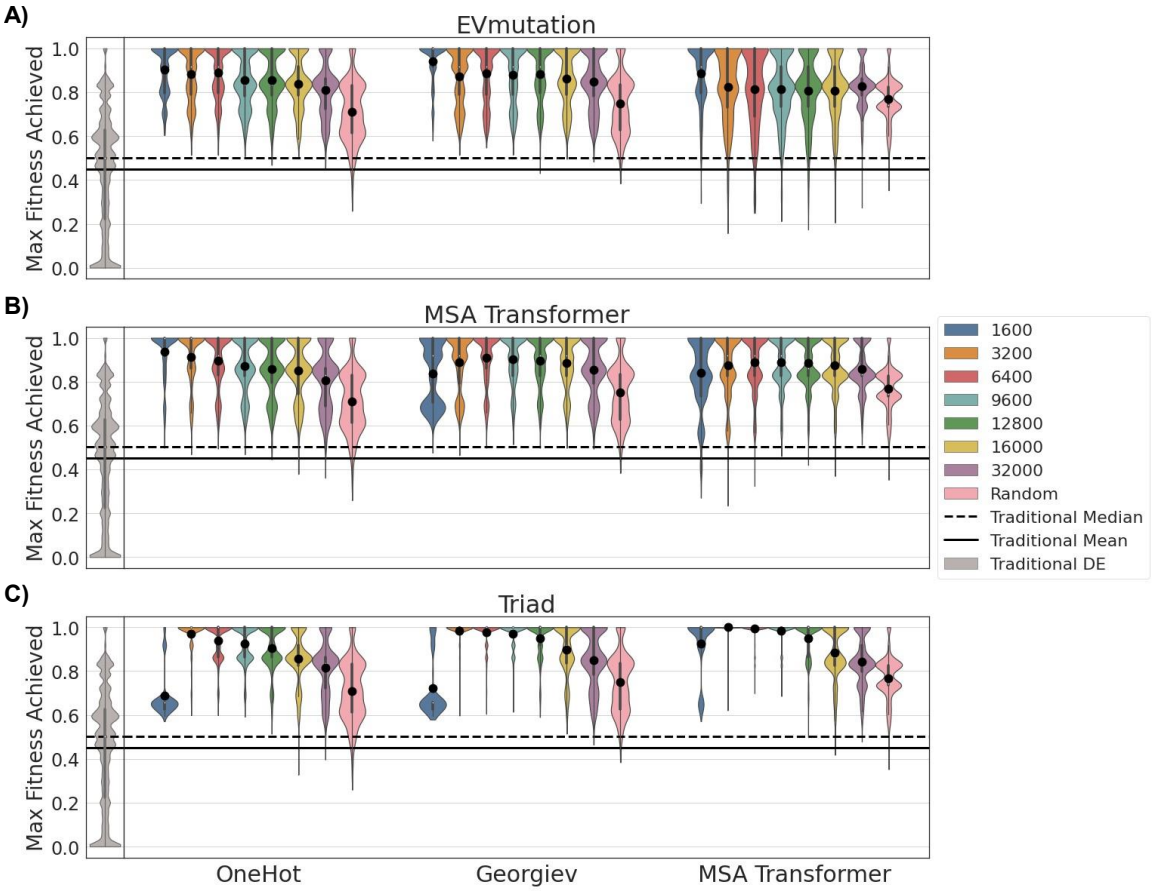


Figure 2-5. Zero-shot prediction for training set design enables highly effective ftMLDE on the GB1 landscape, as measured by maximum fitness achieved in simulated experiments. Each subplot (A–C) shows the effect of different zero-shot predictors on the maximum fitness achieved in simulated ftMLDE experiments. Each violin (except for the grey ones corresponding to simulated traditional DE) represents data from 2000 simulated experiments where 384 variants were used for training and the top 96 predictions were tested. The major groupings of violins within each subplot correspond to different encoding strategies (one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer). The color of each violin corresponds to the zero-shot sampling threshold (i.e., the number of best-ranked variants according to a zero-shot predictor from which random samples were drawn to generate training data). Results of ftMLDE are compared to the results of simulated traditional DE (at the left of each plot, in grey) and standard MLDE (the three pink violins in each plot). (A) The maximum fitness achieved by simulated ftMLDE when EVmutation was used as the zero-shot predictor for training set design. (B) The maximum fitness achieved by simulated ftMLDE when a mask filling protocol using the MSA Transformer was used as the zero-shot predictor for training set design. (C) The maximum fitness achieved by simulated ftMLDE when predicted $\Delta\Delta G$ was used as the zero-shot predictor for training set design. See also, Figure A-6 – Figure A-8 and Data S3.

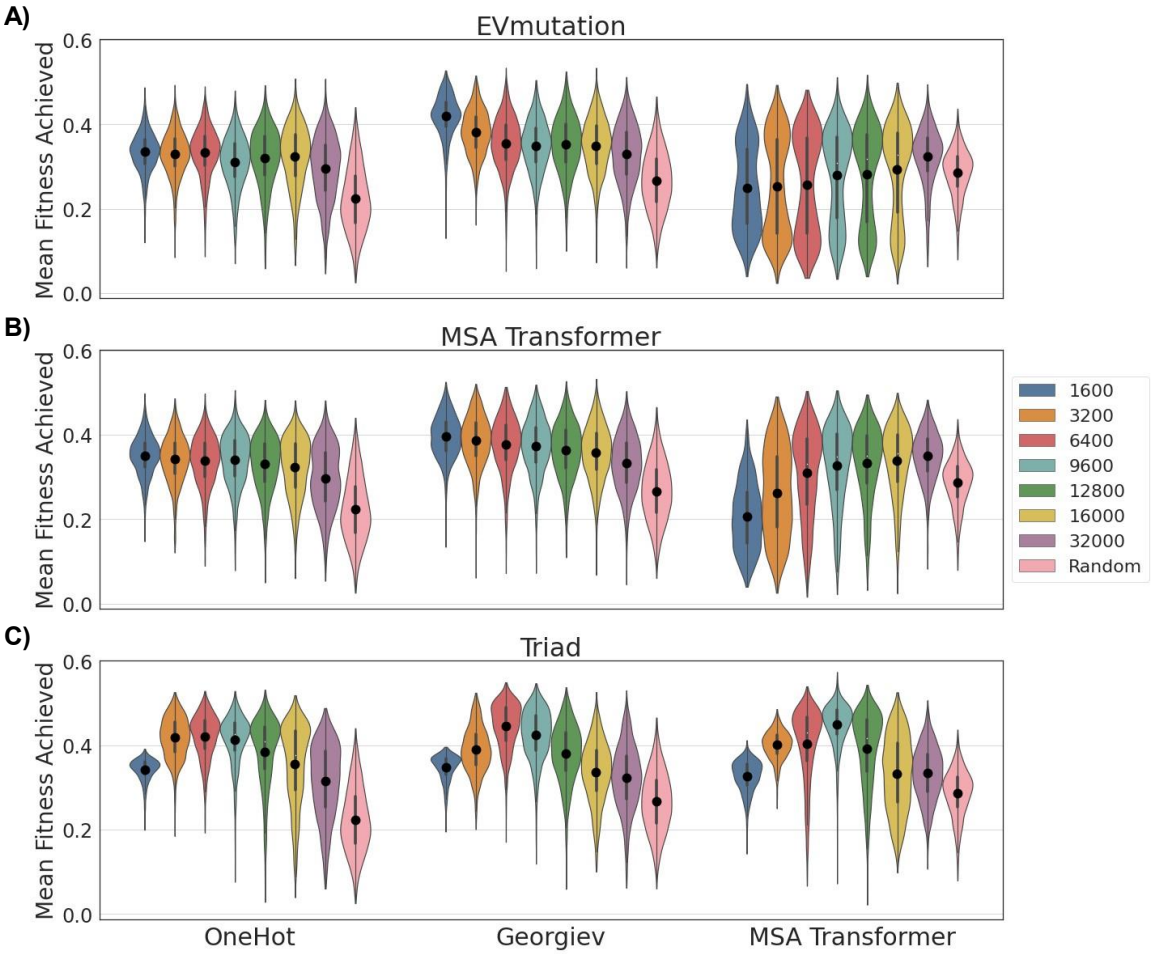


Figure 2-6. Zero-shot prediction for training set design enables highly effective ftMLDE on the GB1 landscape, as measured by mean fitness achieved in simulated experiments. Each subplot (A–C) shows the effect of different zero-shot predictors on the mean fitness achieved in simulated ftMLDE experiments. Each violin represents data from 2000 simulated experiments where 384 variants were used for training and the top 96 predictions were tested. The major groupings of violins within each subplot correspond to different encoding strategies (one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer). The color of each violin corresponds to the zero-shot sampling threshold (i.e., the number of best-ranked variants according to a zero-shot predictor from which random samples were drawn to generate training data). Results of ftMLDE are compared to the results of standard MLDE (the three pink violins in each plot). (A) The mean fitness achieved by simulated ftMLDE when EVmutation was used as the zero-shot predictor for training set design. (B) The mean fitness achieved by simulated ftMLDE when a mask filling protocol using the MSA Transformer was used as the zero-shot predictor for training set design. (C) The mean fitness achieved by simulated ftMLDE when predicted $\Delta\Delta G$ was used as the zero-shot predictor for training set design. See also, Figure A-6, Figure A-9, and Figure A-10 and Data S3.

2.2.9 MLDE Software Enables Wet-Lab Application

To facilitate further development of ftMLDE, as well as to allow for its practical wet-lab application, I developed the MLDE software package, available on the Arnold Lab GitHub (https://github.com/fhalab/MLDE). This repository contains Python scripts for (1) performing zero-shot calculations using EVmutation, DeepSequence, and mask filling using all models from ESM, ProtBert, and ProtBert-BFD; (2) generating encodings for any combinatorial library using one-hot, Georgiev parameters, embeddings from any model in ESM (including those not used for encoding in this work), and embeddings from ProtBert and ProtBert-BFD; and (3) performing ftMLDE as described in this work using any encoding strategy (made available by the MLDE repository or otherwise). The software package was designed for use by non-computational and non-ML experts and can be executed with a simple command line call—all that is required for execution is a fasta file (or an .a2m/.a3m file for procedures using MSAs) with the parent protein sequence and a csv file of combination-fitness data for training.

2.3 Discussion for Chapter 2

I have demonstrated improvements to MLDE that, all together, can make it more efficient than the lowest-possible-screening-burden form of DE for navigating an epistatic, hole-filled, combinatorial protein fitness landscape. While incorporation of more informative encodings and models/regression strategies more amenable to combinatorial protein fitness landscapes was shown to improve MLDE outcome somewhat, by far the greatest improvement came from training set design. Specifically, I showed that a focused training MLDE (ftMLDE) strategy that uses some type of predictor to avoid minimally informative extremely-low-fitness variants in the training data is typically more capable than standard MLDE at identifying the most-fit variants in a combinatorial landscape. From simulated experiments, I noted that the predictor used for training set design in ftMLDE does not need to be capable of identifying particularly high-fitness variants—in tests run using training data purposefully enriched in fitness, eliminating holes had a larger effect on outcome than subsequently raising training data mean fitness. The ability of the predictor to identify diverse sequences, however, is important for improving the probability of identifying the global maximum of a

combinatorial landscape. This concept was best highlighted when using predicted ΔΔG of protein stability as a zero-shot strategy for building training sets, where a balance between sequence diversity and sequence fitness in the training data proved important for maximizing ftMLDE effectiveness (Figure 2-5C, Figure 2-6C, Figure A-6). It is also worth noting that there appears to be an interplay between zero-shot and encoding strategies used and ftMLDE effectiveness, with some combinations of zero-shot predictor and encoding strategy underperforming an MLDE control.

There is, of course, no guarantee that the zero-shot strategies found to be successful for GB1 would be effective for other proteins or other functions. The use of a sequence-based zero-shot strategy, for instance, assumes that the target fitness is well represented by evolutionarily optimized fitness, which will not be the case for all protein engineering problems. Likewise, use of a strategy like predicted ΔΔG assumes that stability (or, potentially, structural conservation) plays a role in fitness determination. In general, the optimal training set design strategy will depend on the protein,[155] and while I have mainly discussed unsupervised zero-shot strategies (i.e., those working off protein sequence or structural data alone) in this chapter, alternate strategies can be imagined. For instance, if a protein scaffold has been used in previous protein engineering studies, a crude non-computational approach would be to avoid mutations that previously destroyed protein function. More robustly, a transfer learning approach could be taken, where an ML model trained using information from related experiments (e.g., evolution of the same protein for a different task, evolution of a different protein for the same task, or even data from previous rounds of MLDE at different positions) is used to predict the effects of mutations in the present experiment.[156] Perhaps even more effectively, fitness information from single-site saturation mutagenesis or error-prone PCR random mutagenesis libraries could be used to predict the fitness of combinations. Indeed, Biswas et al., Hie et al., and Hsu et al. each recently demonstrated approaches where ML models trained on single-site or random mutation data were capable of predicting the fitness of combinations of those mutations.[63,64,79] The use of Gaussian processes in the application of Hie et al. is particularly interesting, as it enables use of the upper confidence bound

algorithm to explicitly balance exploration and exploitation, thus providing a more principled way to inject sequence diversity into training set design while maintaining high fitness.[79,157]

Whatever training set design approach is taken, I would expect its impact on the outcome of ftMLDE to be specific to the shape and makeup of the fitness landscape. For instance, on a non-epistatic landscape, minimalistic traditional DE will deterministically reach the global (and only) fitness maximum; in this case, ftMLDE could at best perform as well as traditional DE regardless of the training set design strategy used (though it may still be able to do so with a lower screening burden). Similarly, as the number of holes in a landscape increases, the probability of a random draw returning primarily uninformative zero-fitness variants increases, and so implementation of an effective training set design strategy will have a greater impact. Thus, the effectiveness of ftMLDE will vary as a function of the shape of the landscape, the number of holes in the landscape, and the availability of robust training set design strategies. It cannot be expected that ftMLDE will always outcompete traditional DE.

Thorough evaluation of the effectiveness of ftMLDE will only be possible once more combinatorial landscape data beyond that provided by the GB1 landscape become available, and since completing this project I have taken part in an effort to build many more of them. Until ftMLDE is tested with those landscapes, however, for now it can be concluded that ftMLDE can be used on combinatorial landscapes known to be highly epistatic and that either contain few holes or else for which confident training set design strategies can be employed. The strategies, concepts, and technology presented in this chapter will serve as a foundation for further evaluation of the generalizability of different encodings, model architectures, regression strategies, and training set design strategies for ftMLDE on combinatorial fitness landscapes. By achieving the GB1 global maximum up to 99.70% of the time with a total screening burden of 480 protein variants, or up to 9.95% of the time with a screening burden of just 80 variants, the ftMLDE protocol presented here outcompeted both traditional DE—which achieved the global optimum just 1.23% of the time—and the original MLDE implementation—which achieved the global optimum 8.17% of the time with a screening burden of 570 variants.[82] This work thus presents a large advance and is, to the best of my

knowledge, the first proven example of a machine learning approach directly outcompeting minimalistic DE. Given the degree to which ftMLDE outcompetes traditional DE on the GB1 landscape, I hope for many more examples to come.

2.4 Financial Support for Chapter 2

I thank NVIDIA Corporation for donation of two Titan V GPUs used in this work and Amazon.com Inc. for donation of AWS computing credits. This work was supported by the NSF Division of Chemical, Bioengineering, Environmental and Transport Systems (CBET 1937902) and by an Amgen Chem-Bio-Engineering Award (CBEA).

Chapter 3

AN EXPLORATION OF SEMI-SUPERVISED MACHINE LEARNING-ASSISTED PROTEIN ENGINEERING STRATEGIES WITH SPICE

Material from this chapter appears in **Wittmann, B. J.**; Johnston, K. E.; Wu, Z.; and Arnold, F. H. (2021) Advances in Machine Learning for Directed Evolution. *Curr. Opin. Struct. Biol.* *69*, 11–18. https://doi.org/10.1016/j.sbi.2021.01.008.

Abstract

Recent efforts in machine learning-assisted protein engineering (MLPE) have focused on developing strategies that can augment small labeled datasets with information extracted from large unlabeled ones, a strategy generally known as "semi-supervised learning." So far, semi-supervised strategies have seen mixed effectiveness when used for protein fitness prediction. In this chapter, I hypothesize that this limited effectiveness results from the failure of current strategies to represent information extracted from unlabeled data in a useful way. This hypothesis is then tested in an exploratory study of semi-supervised MLPE using a model that I designed to represent information extracted from unlabeled protein data in a way that should be *explicitly* informative for downstream learning. Through extensive evaluation using a number of benchmarking tasks, this model, which I call "SPICE" ("Summarizing Proteins using Informed-by-Contact Embeddings"), was found to perform comparably to existing models in a semi-supervised setting. While this conclusion was disappointing, the experiments performed to arrive at it were not for naught, as they strikingly confirmed the limited effectiveness of existing semi-supervised strategies for protein fitness prediction. Specifically, they showed that existing strategies tend to yield comparable if not worse results than fully supervised approaches. Further, barring a single exception when using SPICE, the effectiveness of semi-supervised strategies appeared to be only marginally correlated with the amount of unlabeled data employed, indicating that information contained in those data is not being used effectively. Overall, the work presented in this chapter demonstrates there is still much to be learned about how to most effectively perform semi-supervised MLPE.

3.1 Background and Motivation for Chapter 3

Protein engineering has historically been dominated by two philosophies: "rational design," which attempts to predict the most beneficial mutations using a computer model based in physical and chemical principles,[158] and "directed evolution," which proceeds through rounds of mutagenesis and screening of protein variants in a manner akin to artificial and natural selection.[9,28] Recent years have seen the rise of a third approach to protein engineering—machine learning-assisted protein engineering (MLPE)—that attempts to combine the strengths of both directed evolution and rational design.[16,17,22,24] Just like rational design approaches, the goal of MLPE is to build a predictive model of protein fitness; unlike rational design, however, the models are not built from physical and chemical principles but instead from patterns found in training data. The use of these "black box" predictive models gives MLPE the generalizability of directed evolution while reducing the need for extensive laboratory characterization of protein variants because, just like with rational design, the models can be used to identify the best variants *in silico*.

As with all machine learning (ML) applications, the more high-quality data used to train MLPE models, the more effective those models will be in downstream tasks. The most effective MLPE models will be trained using labeled protein data (i.e., data consisting of protein sequences paired to some measurement of protein fitness);[63,67,68,70] however, as discussed throughout this thesis, these data are often prohibitively expensive to collect in large quantity. In contrast, drastic reductions in sequencing costs have led to a deluge of unlabeled sequence data, and hundreds of millions to billions of protein sequences are now stored in online databases.[15,39–42] As a result, recent research efforts have shifted to developing strategies that can augment small labeled datasets with information extracted from large unlabeled datasets, a strategy generally known as "semi-supervised learning."[22,37,38,46,130]

When used in MLPE, semi-supervised learning consists of an unsupervised learning phase where models are trained on unlabeled data followed by a supervised learning phase where models are trained on labeled protein-fitness data. At the highest level, the goal of supervised

learning is to learn a mapping between an input space (e.g., proteins) and an output space (e.g., protein fitness). If the relationship between those two spaces is simple, then this mapping can be easily learned using a simple model (a model with few learnable parameters) and a small amount of data; however, more complex relationships require more complex models and more data. Before a protein can be passed into a supervised ML algorithm, it must first be numerically encoded, and the encoding scheme chosen defines the structure of the input space. By extension, then, a protein encoding scheme with a simple relationship to protein fitness can reduce the data requirements of supervised MLPE. Indeed, the goal of the unsupervised stage of the semi-supervised pipeline is to learn a numerical representation of protein sequences from unlabeled data that has a simple relationship to protein fitness, thus reducing the need for labeled data in the subsequent supervised stage.

The unsupervised stage of semi-supervised learning—which is also commonly referred to as "unsupervised pretraining" because it occurs before the supervised stage—works on the assumption that all sequenced proteins follow some set of biophysical and evolutionary rules that allow those proteins to be produced and carry out a biological function.[22,37,38] The goal of unsupervised pretraining is to train models to learn the sequence constraints that result from these rules, then represent them in a continuous vector encodings known as "learned embeddings." These learned protein embeddings are assumed to define the relationships between proteins within the context of learned sequence constraints, passing valuable information about what defines a functional protein into the downstream supervised task and, in principle, constituting a highly informative protein encoding scheme.

An early study using semi-supervised MLPE seemed to suggest that learned protein embeddings could be used to drastically reduce the amount of labeled data needed for effective supervised learning with protein sequence-fitness data.[64] Follow-up work has since tempered the initial excitement, however, showing that the effectiveness of learned embeddings is highly context specific, with the use of even the simplest encoding strategies often providing comparable if not superior results.[62,63,65] Indeed, in my own thesis work (see

Chapter 2) I have observed that use of learned embeddings will not always improve supervised learning performance.[33]

Exactly why semi-supervised MLPE is more effective in some situations than others remains unclear. One possibility, however, is that the models used for the unsupervised learning phase (and the embeddings that they produce) do not capture or else do not clearly represent the information critical for predicting protein fitness. With some exceptions,[159,160] most models and training procedures used for unsupervised pretraining are directly adapted from natural language processing (NLP) and have minimal to no biological inspiration.[49,51,60,61,128,131,161,52–59] For instance, models based around the Bidirectional Encoder Representations from Transformers (BERT) architecture—which were originally designed for NLP tasks—have so far been the most popular and effective for the unsupervised pretraining stage of semi-supervised MLPE.[51,55,57,58,61,128,131,161] These large models are described by millions (or billions) of parameters and operate by performing successive mathematical transformations to input data; when used in MLPE, they are trained to reconstruct corrupted protein sequences using either a "masked token prediction" or "next token prediction" strategy.[47] In masked token prediction, the model is fed protein sequences with some percentage of amino acids obscured ("masked"); then, using the context of the unmasked amino acids, the model must predict the identities of the masked ones. Next-token prediction proceeds similarly, only here the model is fed the beginning (or the end) of a protein sequence and tasked with autoregressively predicting the subsequent amino acids. Protein embeddings produced from these models are typically derived from the last layer (the output of the last mathematical transformation) prior to the output layer (the layer that produces predictions of masked amino acids). Because this layer feeds into the output layer, it is assumed that embeddings derived from it must represent information about the rules dictating which amino acids are and are not allowed at given positions; after all, if that information were not present, then the model would be unable to accurately perform next token or masked token prediction. Thus, by using this layer for encoding, information regarding how well a given protein sequence obeys learned sequence constraints can be passed into a downstream task.

Notably, the embeddings derived from models trained with masked token and next token objectives are not designed to explicitly represent information relevant to protein fitness. These training objectives do not require the model to structure the learned embedding space such that it has a simple relationship to protein fitness, so even though the rules governing sequence constraints must be contained in these embeddings, they may not be easily recognizable to a simple supervised model trained on limited data (see Figure 3-1 for an example of an embedding space that explicitly represents determinants of protein fitness).

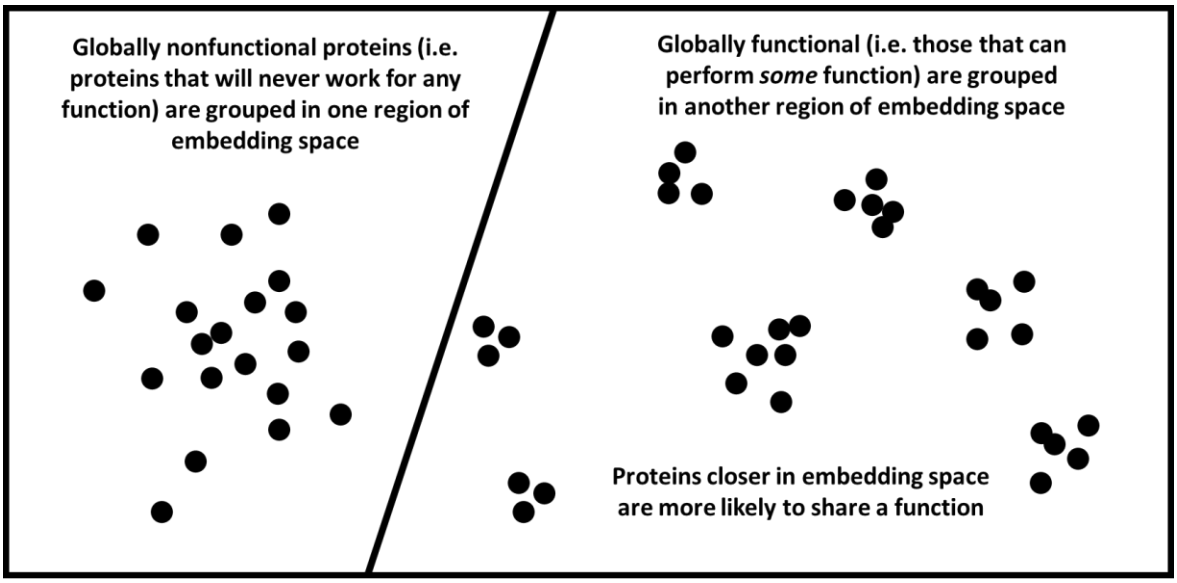


Figure 3-1. A simple example of an explicitly informative embedding space for machine learning-assisted protein engineering. Such an embedding space would clearly distinguish proteins that will never be functional (i.e., due to being unstable) from those that have the potential to be functional. Ideally, such a space would also group proteins with similar fitness levels (the ability to perform a given task) near one another. An implicitly informative space may contain the same information as this explicitly informative space, but would be structured such that extracting that information would be more challenging—the geometric relationship between the embedding space and protein fitness would be more complex.

This chapter is focused on testing whether unsupervised models trained to *explicitly* capture and represent information relevant to protein fitness will produce more effective learned embeddings than those produced by the current models trained to *implicitly* capture and represent that information. Specifically, in this chapter I detail a project I performed during an internship at Microsoft where I attempted to build an unsupervised model that produced

learned embeddings structured to represent known global determinants of protein fitness. In the next section, I detail the motivation and architecture of my biologically inspired language model, which I call "SPICE" ("Summarizing Proteins using Informed-by-Contact Embeddings"). Then, I present and discuss results from a number of benchmarking experiments using embeddings derived from it.

3.2 An Overview of SPICE

This section prevents a theoretical overview of SPICE, focusing in particular on the strategies and data used to encourage it to produce explicitly informative protein embeddings.

3.2.1 Designing an Explicit Embedding Space with SPICE

As discussed in the last section, BERT-style transformer models have, to date, been the most effective models for semi-supervised MLPE.[51,55,57,58,61,128,131,161] When used to produce embeddings, these "protein-BERT" models will take a protein sequence as their input and output a matrix of amino acid embeddings. For a protein of length L, this means that a protein-BERT model will output a matrix of shape L × D, where "D" is the dimensionality of the embedding.[h] Most MLPE tasks are performed on the protein level, however, not the amino acid level, so this output matrix must be reduced to a vector before it is used in a downstream supervised task. The most common strategy for this reduction is "mean-pooling," where the mean of the output matrix is taken over the amino acid axis to produce a "global" embedding vector of length D. While simple and easy to apply, this strategy is completely biologically uninspired. For instance, we would expect that positions in an enzyme active site would be more relevant to representing the determinants of protein fitness as a whole, so it may be reasonable to give their associated embeddings more weight when pooling to the global vector.

Recognizing the potential limitations of mean-pooling, alternate strategies have been developed that attempt to learn ideal operations for reducing the amino acid-level matrix to a protein-level representation, distinctly weighting the contributions of different features and

[h] The dimensionality of the embedding corresponds to the number of learned features used to represent each amino acid.

amino acids when building the global vector. Unfortunately, however, almost all of these strategies are applied during the supervised learning stage of the semi-supervised workflow.[58,162] Considering that the supervised stage of semi-supervised MLPE is often performed in the low-N setting (i.e., with limited labeled training data), this approach runs the risk of overfitting to the training set and losing generalizability in the downstream application. Indeed, Detlefsen et al. have shown that learning a global representation of protein sequences during the unsupervised pretraining stage of the semi-supervised workflow can be beneficial to downstream supervised learning.[66]

Just like the strategy used by Detlefsen et al., the core concept of SPICE is to learn the global representation from unlabeled data during unsupervised pretraining. Unlike Detlefsen et al., however, who used a pooling strategy that was not biologically inspired,[i] the SPICE learning objective was designed to explicitly represent global determinants of protein fitness in its embedding space. A global determinant of protein fitness is a characteristic shared by all proteins, regardless of function, that can be used to predict whether a protein will be fit. Protein stability is among the most obvious and intuitive such global determinants, as proteins that are not stable will not be able to hold the 3D structure that grants a function. SPICE is trained such that the distance between two global protein embeddings should be correlated with the change in stability between the represented proteins (the $\Delta\Delta G$ of stabilization when mutating one to the other), aiming to construct an embedding space where unstable proteins are grouped together in one region of embedding space and stable ones are grouped in another. Such a layout provides an explicitly informative embedding space like that depicted in Figure 3-1 because proteins that are embedded in the "unstable" region can be easily assigned a fitness of "0." All that a downstream supervised model must do is identify this unstable region from labeled training data, something that should be simple to accomplish with just a few training examples.

[i] Specifically, the model used by Detlefsen et al. (2022) was trained to reconstruct the original protein sequence from a pooled representation, just like in an autoencoder.

3.2.2 Contact Disruption Upon Mutation as a Heuristic for ΔΔG of Stabilization

As discussed in more detail in Section 3.2.3, the core component of the SPICE model is a 43 million-parameter transformer encoder. A large amount of data is needed to effectively train models of this size, so training SPICE to directly represent protein stability in its embedding space would require many millions of datapoints capturing ΔΔG of stabilization between related protein pairs. Unfortunately, there is no such dataset of this size, and building one would be impractical, especially when considering that the goal of SPICE and other semi-supervised learning strategies is to minimize the need for additional data collection in the first place.

As a heuristic for ΔΔG of stabilization, SPICE instead uses "contact disruption upon mutation" (hereafter referred to as "contact disruption"), which is defined here as the number of contacts in a protein's 3D structure that are broken upon mutation, where a "contact" is simply an interaction between two amino acids. For instance, if position 31 in a protein sequence contains an alanine and this alanine is in contact with positions 45 and 51 in the 3D structure, mutating that alanine to any other amino acid would break two contacts (contacts 31–45 and 31–51). Importantly, all that is needed to calculate contact disruption is a protein structure and a list of candidate mutations; from here, it can be computationally derived by simply mapping the candidate mutations on to a structure and counting the number of contacts broken. As a result, unlike ΔΔG of stabilization, millions of contact disruptions can be computed in seconds, producing more than enough data for training SPICE.

Generally speaking, the more contacts that are broken upon introduction of mutations to a protein (the higher the contact disruption), the more destabilizing we expect those mutations to be and the less likely the mutated protein is to share a structure with the original. Thus, if we train a model to embed proteins such that the distance between global embeddings encodes contact disruption, the distance between two embedding vectors will reflect both the change in stability between the two proteins they encode as well as how structurally similar they are. Because structure ultimately determines protein function, an embedding space that orders proteins by structural similarity should also encode local determinants of fitness.

Specifically, we would expect that proteins with a smaller contact disruption are more likely to show similar fitness levels and vice versa. In many ways, then, a contact disruption-based embedding space provides more information than a ΔΔG of stabilization-based embedding space by also encoding the degree of functional difference we might expect between two stable proteins. Just like in the example of an explicitly informative embedding space presented in Figure 3-1, using contact disruption as a heuristic for ΔΔG of stabilization allows for construction of an embedding space that captures both global and local determinants of protein fitness.

3.2.3 The SPICE Architecture

An overview of the SPICE architecture is given in Figure 3-2A. The core of the model is a transformer encoder derived from the evolutionary scale modeling (ESM) package.[55] After the transformer core, a pooling layer based on the attention-weighted mean proposed by Rao et al. is used to generate global embeddings.[58] Various projection layers map from the global embedding to calculate SPICE training losses, all of which are described in Section 3.2.5. It is these training losses that encourage SPICE to learn an explicitly informative embedding space.

An alternate form of SPICE—variational SPICE—takes inspiration from variational autoencoders (VAEs) to encode global embeddings as multivariate normal distributions rather than defined vectors. This alternate model architecture allows for more probabilistically grounded comparisons of embedded proteins; it also allows an inherent sense of uncertainty to be encoded in the embedding space. Additional details on training variational SPICE and its applications are provided in later sections.

3.2.4 SPICE Training Data and Calculation of Contact Disruption

Two sources of data are used to train SPICE. The first is a set of ~49 million redundancy-reduced protein sequences from the UniRef50 database that comprehensively covers the full space of known proteins.[42] The second is the trRosetta dataset, which consists of ~30,000 protein structures each paired to a multiple sequence alignment (MSA) of homologous

(evolutionarily related) sequences; the MSA paired to each structure was built using that structure's associated sequence as the query sequence.[163]

The trRosetta dataset is critical for building the explicit representation of contact disruption in the SPICE embedding space. As mentioned earlier, calculation of contact disruption requires both a template protein structure and a list of candidate mutant sequences. To build the contact disruption dataset needed for training SPICE, the structures in the trRosetta dataset serve as template structures and the list of aligned sequences in the MSAs provide candidate mutations. Specifically, for each MSA, a group of mutant sequences is built by transferring mutations from the aligned sequences to the query sequence, ignoring gap characters. The mutant sequences of each group are then mapped on to the structure associated with their MSA and the set of amino acid contacts present in the resultant mutant structures are determined (where contacting residues are defined as those with any non-hydrogen atom within 4.5 Å of one another). Within each group of mutant sequences, pairwise comparisons are then made to determine the number and type of contacts broken and maintained between mutants, providing the information needed for incorporating contact disruption into the SPICE training procedure.

3.2.5 The SPICE Training Procedure

Unlike most protein-BERT models, which are trained using a single training objective, SPICE is trained using a varied set of training objectives, all designed to extract information from unlabeled protein data and represent it in a useful way. Training proceeds by alternating between objectives based on data derived from UniRef50 and objectives based on contact disruption calculations. These objectives include both protein-specific tasks, which are those concerned with predicting some characteristic of a protein in isolation, and contrastive tasks, which are those concerned with representing some relationship between a pair or more of proteins. The motivations behind the different objectives, their specific implementations, and the data they use are all detailed in the below list and Figure 3-2:

1. The first objective is a masked-token prediction protocol that uses the UniRef50 dataset, just like what was described in the introduction. This is trained using the

output for the transformer encoder of SPICE and is a protein-specific, amino acid-level task. It is included to learn the sequence constraints that determine a functional protein. This is the only amino acid-level task; all other tasks are designed to encourage the model to pool and organize the information captured at this step to build an explicitly informative protein-level representation.

2. The next objective is a contrastive task comparing the global embedding vectors of a pair of mutant protein sequences derived from the trRosetta dataset. This objective uses a root-mean-squared error to encourage the Euclidean distance between the two global vectors to be equivalent to the fraction of total contacts broken between the two sequences they represent. In other words, the model is trained such that the distance between the global vectors will reflect the number of contacts broken were one of the input sequences to be mutated to the other. Note that, as with all other contrastive tasks, only mutant sequences derived from the same MSA and structure (see Section 3.2.4 for details) are used for optimizing this objective.

3. The type of contacts broken upon mutation can be important for determining the change to a protein's structure, stability, and, by extension, fitness. For instance, disrupting an alanine–serine contact by mutating the alanine to glycine is less likely to drastically alter the protein structure than mutating to a tyrosine. To encourage SPICE to include information like this in its learned embeddings, another contrastive objective is included that pushes the model to encode the *types* of contacts broken between two mutant proteins (again derived from the trRosetta dataset) in the *direction* of the vector connecting them in embedding space. This is accomplished by (1) calculating the global embedding vectors for a pair of mutant protein sequences, (2) using the vector difference between the two global embedding vectors to predict the distribution of the types of contacts broken when mutating one of the input sequences to produce the other, and (3) minimizing the Kullback–Leibler divergence (KL divergence) between the predicted distribution of broken contacts

and true distribution of broken contacts.[j] It is important to note that the distribution of contacts broken changes depending on the direction of mutation—the contacts broken moving from sequence 1 to sequence 2 are different than those broken when moving from sequence 2 to sequence 1. To handle this asymmetry, steps two and three of this objective are performed twice: once using the difference vector that transforms global embedding 1 to global embedding 2 and again using the difference vector that transforms global embedding 2 to global embedding 1, changing the target distribution of broken contacts as appropriate.

4. Similarly important to the types of contacts broken upon mutation are those maintained. Another contrastive training objective is thus employed that encourages the model to encode the types of contacts maintained between two mutant proteins in the direction of the vector connecting them in embedding space. This is accomplished using the same strategy as in the previous objective, only now predicting the distribution of contacts maintained rather than broken.

5. A final objective used by all SPICE model variations is a protein-specific global loss focused on capturing the types of contacts in the global vectors. This objective proceeds by (1) calculating the global embedding of a *single* mutant sequence derived from the trRosetta dataset, (2) using that global embedding to predict the distribution of the types of contacts in that protein, and (3) minimizing the KL divergence between the predicted and true distributions. The motivation for this objective is to encourage SPICE to place proteins with similar contact profiles near one another in the learned global embedding space. Proteins with similar contact types should share similar structures and generally be expected to perform similar functions, so this objective serves as a proxy for encouraging the model to group proteins of similar function near one another in the learned embedding space.

[j] KL divergence can be thought of as a measure of how well one distribution captures the information contained in another, and so is a natural metric to use when comparing two distributions.

There are also a number of optional tasks that were employed to test slight variations of the SPICE architecture. The optional tasks are all detailed below:

1. As mentioned in Section 3.2.3, an alternate SPICE architecture encodes proteins as distributions in embedding space rather than points. This "variational SPICE" employs the same objectives as regular SPICE but is trained using a protocol inspired by that used to train variational autoencoders (VAEs). Specifically, during training, a vector of means and log variances is learned that describes a multivariate normal distribution. Global embeddings are then drawn from this distribution before being used to optimize the objectives used to train the standard SPICE architecture. As with VAEs, an additional KL divergence loss term is added that encourages distributions learned to be similar to a predefined Gaussian prior.[164] Throughout the work presented in the rest of this chapter, the Gaussian prior always had a mean of zero but the standard deviation was varied.

2. One final optional objective used during SPICE training—whether using the standard or variational SPICE architectures—is a protein-specific objective where an additional decoder model is added to SPICE to attempt to reconstruct the original protein sequence from the global representation. This is only performed using sequences from the UniRef50 dataset; it is not performed using mutant sequences derived from the trRosetta dataset. This was the objective used by Detlefsen et al. in their work showing that it can be beneficial to learn a global representation during unsupervised pretraining.[66] When included, it is done so to better enable comparison between that work and the work presented in the remainder of this chapter.

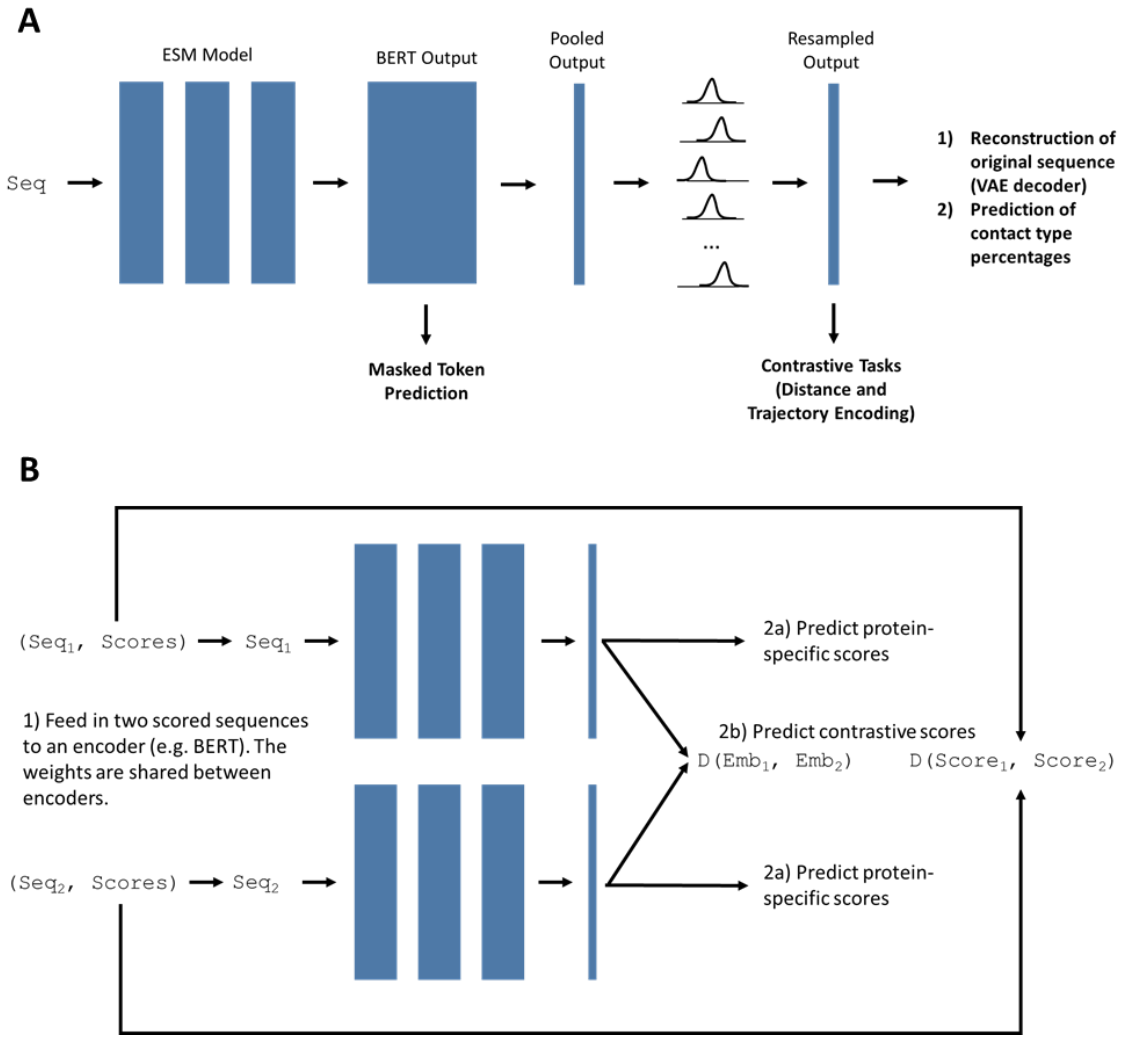


Figure 3-2. The SPICE architecture and training scheme. A) The SPICE model architecture and all possible training tasks. Note that, depending on the model variant, not all training tasks are applied and some transformations within the model are skipped—this figure is meant to depict all options. The core model consists of a six-layer transformer encoder followed by a pooling layer. Optionally, a variational form of the pooling layer is treated as a vector of means and log variances—in such a case, SPICE is referred to as "variational SPICE." Global losses for variational SPICE are calculated using samples from the learned distributions; global losses for non-variational SPICE are calculated using the pooled layer. In other words, resampling is only performed for variational SPICE. B) The SPICE training procedure. SPICE is trained with both protein-specific and contrastive tasks. For contrastive tasks (as depicted in this figure), pairs of sequences with known contacts are passed through SPICE at the same time. Importantly, the model never directly sees the contacts; contrastive losses are calculated using the number broken between the two proteins, however. Non-contrastive tasks are also calculated to train SPICE to capture information relevant to individual proteins in addition to capturing information important for comparing proteins.

3.3 Semi-Supervised Learning with SPICE

The previous section gave a theoretical overview of SPICE and presented general strategies for training it. In this section, I present the results of experiments designed to test the effectiveness of SPICE in semi-supervised MLPE.

3.3.1 Unsupervised Pretraining of SPICE Variants

It should be clear from Sections 3.2.3 and 3.2.5 that there are a number of possible variations on the core SPICE architecture. Most notable are the differences between SPICE and variational SPICE and whether the sequence reconstruction objective is included during training. There are more subtle changes that can be made to the architecture as well, however, that can be expected to have an effect on the utility of SPICE for semi-supervised MLPE. For instance, the dimensionality of the learned global embedding can be tuned to balance the descriptiveness and generalizability of the features learned during unsupervised pretraining.[k] Likewise, when training variational SPICE the standard deviation of the Gaussian prior can be modulated to encode how functionally similar proteins are expected to be *a priori*.[l]

To test the impact of the various design considerations of SPICE on its downstream applicability, I trained nine different model instances following the general training procedures detailed in Section 3.2.5. Three of these were non-variational SPICE instances that differed in the encoding dimensionality tested (128 or 768) and whether or not the reconstruction task was included; the others were all variational SPICE instances that differed in the encoding dimensionality tested (again, 128 or 768) and prior standard

[k] If the dimensionality is high, then the model will have more flexibility in how it represents proteins, allowing it to learn richer, more descriptive features, but perhaps limiting its downstream applicability to regions of sequence space not seen during unsupervised pretraining. Conversely, if the dimensionality is low, then the model must learn more broadly applicable features that, while perhaps not the most descriptive, may be better capable of generalizing to new regions of sequence space.

[l] As described in detail in Section 3.2.2, the distance between proteins in the SPICE embedding space represents how functionally similar they are expected to be. When using variational SPICE, proteins are embedded as distributions rather than vectors, so the level of overlap between distributions can be thought of as representing the probability that two proteins share the same function. The value of the standard deviation on the SPICE Gaussian prior can thus be intuitively thought to encode how similar nearby proteins are expected to be to one another *a priori*: a low value encourages the model to learn narrow distributions that overlap little with others and vice versa.

deviation (0.05, 0.1, 0.2, or 1). All variational SPICE model variants included the reconstruction loss as an objective.

The sections that follow compare the effectiveness of SPICE encodings derived from the nine different model instances. To clearly distinguish each model, I refer to them using a four-component hyphen-delimited naming convention that clearly describes their different designs. The first section of each name gives whether or not the model is a variational SPICE instance: "Variational" is used for variational instances while "NoVariational" is used for the others. The second section gives whether a reconstruction term was applied to the model: "Reconstruction" indicates that a reconstruction term was used during training while "NoReconstruction" indicates that one was not. The third section gives the embedding dimensionality. The fourth section gives the prior standard deviation used for variational models: "s005" means that the prior standard deviation was 0.05, "s01" means that it was 0.1, "s02" means that it was 0.2, and "s1" means that it was 1. The non-variational instances do not consider a prior standard deviation during training, so the last section of the name can be ignored for them.

3.3.2 SPICE Embeddings for Supervised Learning

Fair comparison of different semi-supervised MLPE approaches requires use of what are known as "benchmark tasks." Designed to reflect real-world MLPE applications, these tasks most commonly consist of preset training data (which is used to train the ML model) and testing data (which is used to evaluate the performance of the trained model) derived from larger labeled datasets; critically, they provide standardized conditions for comparing newly proposed semi-supervised models, training procedures, etc. To evaluate the effectiveness of embeddings derived from SPICE, I gathered labeled datasets consisting of protein sequence-fitness data, training indices, and testing indices needed for performing three benchmark tasks. Two of these tasks—one to do with predicting fluorescence of green fluorescent proteins (GFPs) and another with predicting protein stability—were derived from the TAPE

GitHub repository, while the third—to do with predicting the viability of adeno-associated virus (AAV) capsid proteins[m]—was derived from the FLIP GitHub repository.[58,73,118,148,165]

For each of the three collected benchmark tasks I tested three different predictive strategies using SPICE embeddings derived from the various model instances described in Section 3.3.1. The motivation behind each of these strategies along with their implementation details are in the list below:

1. The first strategy trained an unregularized linear model to map learned global embeddings to fitness labels. As mentioned multiple times, the goal of SPICE is to structure the learned representation space such that there is a simple relationship between protein embeddings and protein fitness that can be readily used to inform downstream supervised learning. Linear models are the simplest models that are commonly used in the supervised stage of semi-supervised MLPE and so act as a natural baseline model to include when testing the effectiveness of SPICE. Note that the learned vector of mean and log variances was used as the embedding when variational SPICE model instances were used for this strategy, not a global vector sampled from the distribution defined by it.

2. Just because a linear model is simple does not necessarily mean that it would be ideal for modeling a simple relationship between learned embeddings and fitness—simple relationships can be nonlinear. For instance, a situation can be imagined where proteins are no longer functional beyond a certain number of broken contacts from any functional variant (e.g., due to those functional variants being marginally stable). In the idealized SPICE embedding space, such a relationship would manifest as a central island of functional sequences surrounded by a hypersphere of nonfunctional ones, constituting a highly nonlinear, yet still simple relationship between learned embedding space and fitness. Nonlinear relationships like these were accounted for

[m] The tasks used from the TAPE repository are those presented in its accompanying publication. The task from the FLIP repository is, by contrast, from a pre-publication draft of the repository: specifically, “natural task 1” from commit “6737e79.”

by the second predictive strategy, which was applied to the variational SPICE instances. Specifically, for each test set member, a score correlated with fitness was calculated as a weighted sum of training set fitnesses where, for each training set member, the weight applied was proportional to the degree of overlap between its learned distribution and the distribution of the test set member as measured by the Bhattacharyya coefficient. Importantly, this "from-distribution" prediction strategy has no learnable parameters, making it as simple a model as possible. For it to be effective, the determinants of fitness must thus be explicitly represented in the SPICE embedding space, with proteins with similar fitness grouped near one another.

3. The final predictive strategy was a simple voting ensemble where the ranks of each test set member predicted by the first two strategies were averaged to build a composite score. As with most ensembling strategies, the idea behind this approach was to balance the relative strengths and weaknesses of the component models. For instance, when the linear and from-distribution approaches agree, we can be more confident in the predictions—this confidence is reflected in both models assigning a high rank and so the ensemble average returning a high rank. By contrast, when the two component models disagree, the average rank returned by the ensemble will fall between them, reflecting lower confidence in either of the results.

The results from applying these predictive strategies to the three benchmark tasks are shown in Figure 3-3 – Figure 3-5, where each column in these figures corresponds to the results from a different SPICE variant. The y-axes of these plots show predictive performance as measured by the Spearman rank correlation coefficient (Spearman ρ) calculated between predicted and true values of the test data; the x-axes show the degree of unsupervised pretraining, with results at the 0^{th} epoch calculated using embeddings derived from a randomly initialized model and results at later epochs calculated using embeddings from models trained on subsequently more data. Also present in these figures are two baselines calculated using the BERT-style ESM model making up the core of the SPICE architecture: the black horizontal line in each figure provides benchmarking results using the published

form of this model (i.e. using the weights learned in the work of Rives et al.);[55] the additional columns present in Figure 3-3 and Figure 3-4 provide benchmarking results using an ESM model retrained with a masked-token prediction objective on the exact UniRef50 data used to train SPICE. Both ESM baselines were calculated by training a linear model to relate mean-pooled learned embeddings to protein fitness, the current most common practice for semi-supervised MLPE. Because the ESM model provides the core of SPICE and was originally trained with a masked-token prediction objective, it acts as an excellent state-of-the-art baseline against which SPICE can be compared.

From Figure 3-3 – Figure 3-5, it is clear that most of the design considerations discussed in the previous section appear to have minimal if any impact on the effectiveness of SPICE embeddings, at least on the benchmark tasks tested here. Specifically, use of variational SPICE versus non-variational SPICE, the choice of the standard deviation for the Gaussian prior of variational SPICE, and whether or not a reconstruction term was used all had no noticeable effect on predictive performance. Only the embedding dimension appeared to have some, albeit minor, impact, with models trained using a larger embedding dimension consistently outcompeting otherwise equivalent ones trained with a smaller embedding dimension.

Counter to expectation, and more striking than any of the intended observations, Figure 3-3 – Figure 3-5 also show that there generally appears to be minimal correlation between the extent of unsupervised pretraining and the effectiveness of learned embeddings in downstream tasks, regardless of predictive strategy and embedding-generating model employed. For instance, using the GFP dataset (Figure 3-3), increased pretraining led to no significant gains in embedding effectiveness over the randomly initialized embeddings regardless of model or predictive strategy used. Using the AAV dataset (Figure 3-5), increased pretraining led to marginal improvements at best, but could actually worsen performance for some SPICE model instances. The only exception came from using the stability dataset (Figure 3-4), where even though increased pretraining led to inconsistent and erratic results using the linear predictive strategy, it did in fact lead to continuously improving

performance using the from-distribution predictive strategy. Barring this single exception, however, these results suggest that neither SPICE nor ESM are very effective at building embeddings useful for fitness prediction tasks. Notably, in all cases, the retrained ESM model and at least one SPICE model achieved performance that matched or exceeded both (1) previously reported results using the same benchmark tasks but different embedding-generating models and (2) the results that I calculated using embeddings derived from the published ESM model (the black horizontal line in each figure).[58] Combined with recent work from another team showing a similar result,[161] this strongly indicates that the unexpected relationship between pretraining and predictive performance is not a result of an error made during model training, but instead a genuine (yet surprising) feature of unsupervised pretraining in semi-supervised MLPE.

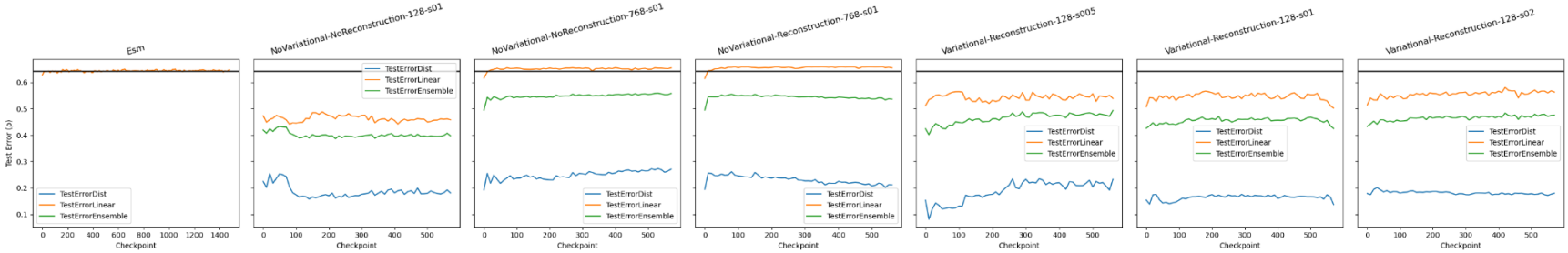


Figure 3-3. Performance of three downstream prediction strategies on the GFP dataset as a function of the number of training epochs (i.e., the amount of unsupervised pretraining). The y-axis in each plot is Spearman ρ and the x-axis is the number of training epochs. Results for six different SPICE variants and both a pretrained and retrained ESM control are shown. The retrained baseline is shown in the leftmost subplot while the pretrained baseline is shown as a black line. Note that the from-distribution (blue) and ensemble (green) calculations are only relevant to the variational SPICE models. The lines corresponding to these prediction strategies in non-variational SPICE plots can be ignored.

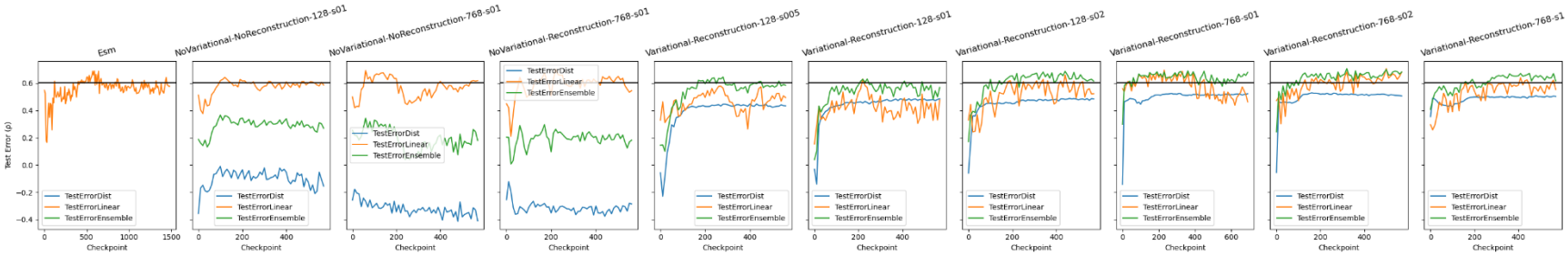


Figure 3-4. Performance of three downstream prediction strategies on the stability dataset as a function of the number of training epochs (i.e., the amount of unsupervised pretraining). The y-axis in each plot is Spearman ρ and the x-axis is the number of training epochs. Results for nine different SPICE variants and both a pretrained and retrained ESM control are shown. The retrained baseline is shown in the leftmost subplot while the pretrained baseline is shown as a black line. Note that the from-distribution (blue) and ensemble (green) calculations are only relevant to the variational SPICE models. The lines corresponding to these prediction strategies in non-variational SPICE plots can be ignored.

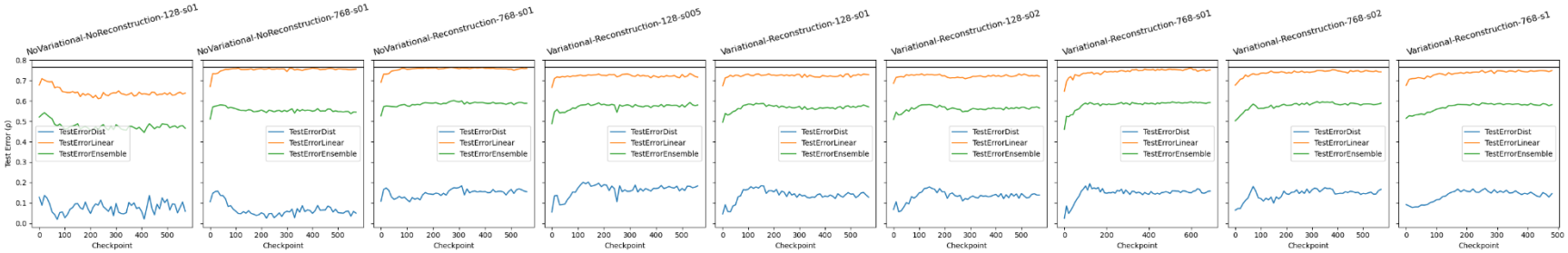


Figure 3-5. Performance of three downstream prediction strategies on the AAV dataset as a function of the number of training epochs (i.e., the amount of unsupervised pretraining). The y-axis in each plot is Spearman ρ and the x-axis is the number of training epochs. Results for nine different SPICE variants and a pretrained ESM control are shown. The pretrained baseline is shown as a black line. Note that the from-distribution (blue) and ensemble (green) calculations are only relevant to the variational SPICE models. The lines corresponding to these prediction strategies in non-variational SPICE plots can be ignored.

It must be noted that semi-supervised learning is expected to be most effective in the low-N case where there are limited labeled data available to train a downstream supervised model. In each of the benchmark tests performed to build Figure 3-3 – Figure 3-5, however, there were abundant labeled training data available for training downstream supervised models. To be sure that the observation that unsupervised pretraining was largely ineffective was not simply a result of using a large amount of training data for the supervised stage of the workflow, I performed additional tests where I subsampled the training split of the AAV benchmark to evaluate the effectiveness of SPICE and ESM embeddings. Specifically, I drew 200 training sets each of 96, 384, and 1536 datapoints; for each of these training sets, I then tested the effectiveness of learned embeddings derived from the nine different SPICE model variants at different stages of pretraining, again using the three predictive strategies described earlier in this section. The results are shown in Figure 3-6 – Figure 3-8 as boxplots over the number of checkpoints rather than the line plots that were used in Figure 3-3 – Figure 3-5. There is little difference between the results in this low-N case compared to the results derived from training downstream supervised models on the full dataset, with the effectiveness of embeddings derived from most SPICE variants seeing minimal improvement with additional pretraining.

Of course, all the above results also demonstrate that embeddings derived from SPICE are at best as useful for semi-supervised fitness prediction as those derived from ESM. There may still be promise in approaches like SPICE, however. As noted earlier, the from-distribution prediction strategy on the stability benchmark task provided a single exception to the general observation that increased pretraining did not consistently improve embedding effectiveness, with the accuracy of the predictions increasing considerably and smoothly with increased pretraining. Even though the from-distribution predictions were not as effective on the stability task as other predictive strategies, the fact that their accuracy consistently improved with increased pretraining is noteworthy, as this suggests a direct relationship between the extent to which unsupervised pretraining objectives are met and the effectiveness of the embeddings learned. This observation is particularly interesting when considering that the from-distribution prediction strategy has no learnable parameters, so any information used

to make predictions is derived solely from information contained in the learned embedding space and the fitness labels. Of course, of the three benchmark datasets tested, we might expect that the embedding space learned by SPICE would be most informative for the stability task: contact disruption was chosen to be modeled by SPICE because it can serve as a heuristic for stability, so it would be reasonable to assume that an embedding space built to represent changes in contact disruption upon mutation would also represent changes in stability upon mutation. This is not so much the point, however; rather, it is that these results suggest that models and unsupervised pretraining objectives designed to explicitly represent determinants of fitness combined with predictive strategies explicitly designed to take advantage of a resultant learned representation can indeed lead to predictable results in semi-supervised MLPE, unlike existing approaches. Thus, even though SPICE may not provide a universal solution for the unsupervised stage of semi-supervised MLPE, and even though the solution that it provides does not surpass existing strategies, it could provide a template from which more effective and predictable semi-supervised MLPE strategies can be developed. That being said, this is a single example, and far more investigation than the exploratory studies performed here would need to be performed to properly test this hypothesis.

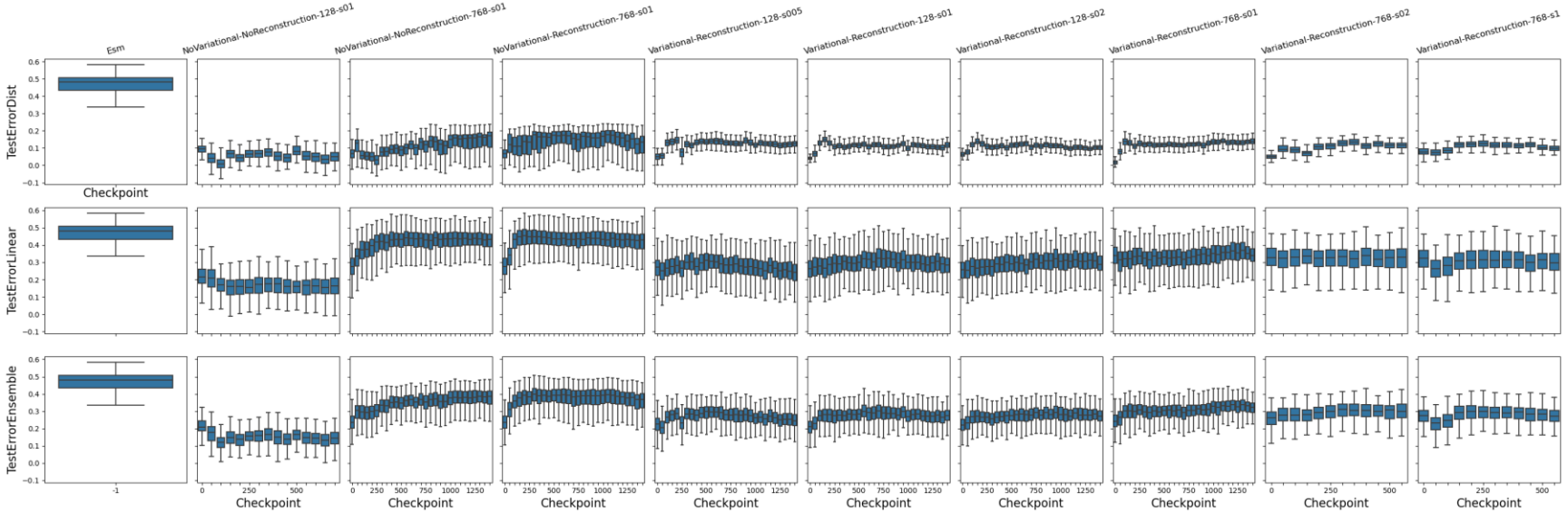


Figure 3-6. Results comparing downstream supervised processes using either SPICE embeddings or ESM embeddings when 96 training examples are used from the AAV dataset. The y-axis is Spearman ρ and the x-axis is the number of training epochs. Results for nine variations on SPICE are presented. The only ESM result presented is from the six-layer pretrained model provided by the ESM repository. The same information as in Figure 3-3 – Figure 3-5 is presented here; only, rather than showing the " from-distribution," " linear," and " ensemble" predictions on the same plot, they are now each presented in a separate row. The first row of plots gives the results from-distribution predictions; the second row is for the linear model; the third row is the ensemble. Note that the plots in the first column are derived from a linear model applied to mean-pooled embeddings from the pretrained ESM model from the ESM GitHub repository—these results are only really comparable to SPICE results presented in the second row. Depending on the specific SPICE model variant tested, results from embeddings derived from ESM are comparable or superior to those derived from SPICE. Note again that, just like in Figure 3-3 – Figure 3-5, the from-distribution (first row) and ensemble (last row) calculations are only relevant to the variational SPICE models.

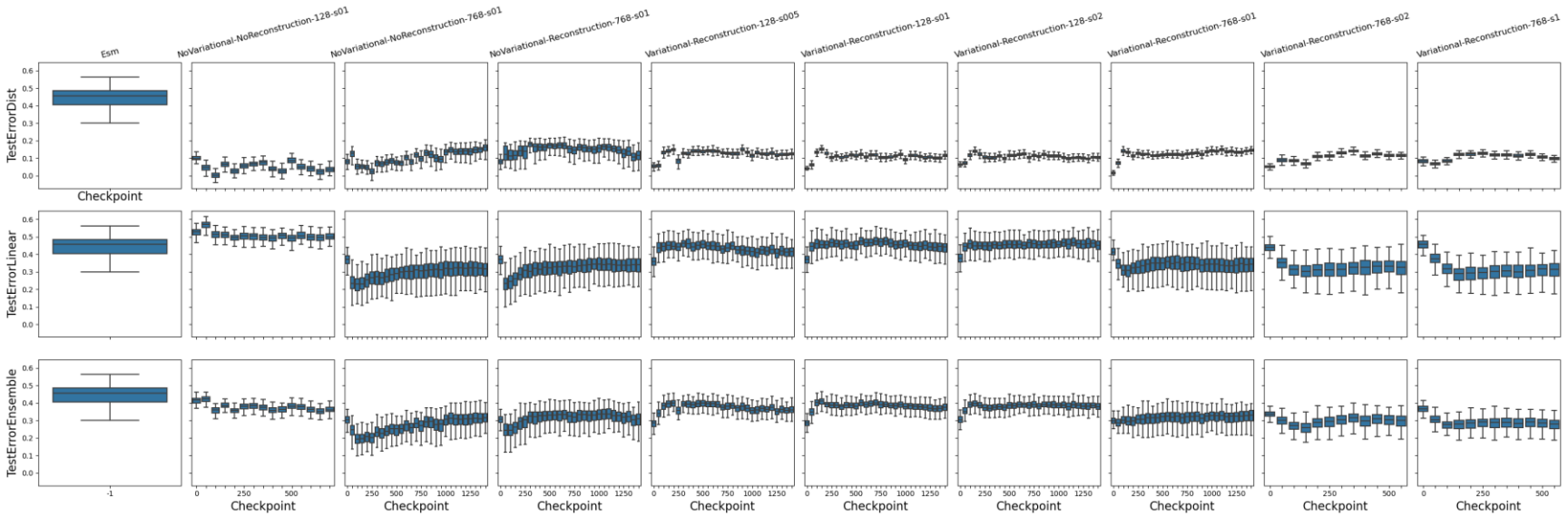


Figure 3-7. Results comparing downstream supervised processes using either SPICE embeddings or ESM embeddings when 384 training examples are used from the AAV dataset. The y-axis is Spearman ρ and the x-axis is the number of training epochs. Results for nine variations on SPICE are presented. The only ESM result presented is from the six-layer pretrained model provided by the ESM repository. The same information as in Figure 3-3 – Figure 3-5 is presented here; only, rather than showing the " from-distribution," " linear," and " ensemble" predictions on the same plot, they are now each presented in a separate row. The first row of plots gives the results from-distribution predictions; the second row is for the linear model; the third row is the ensemble. Note that the plots in the first column are derived from a linear model applied to mean-pooled embeddings from the pretrained ESM model from the ESM GitHub repository—these results are only really comparable to SPICE results presented in the second row. Depending on the specific SPICE model variant tested, results from embeddings derived from ESM are comparable or superior to those derived from SPICE. Note again that, just like in Figure 3-3 – Figure 3-5, the from-distribution (first row) and ensemble (last row) calculations are only relevant to the variational SPICE models.

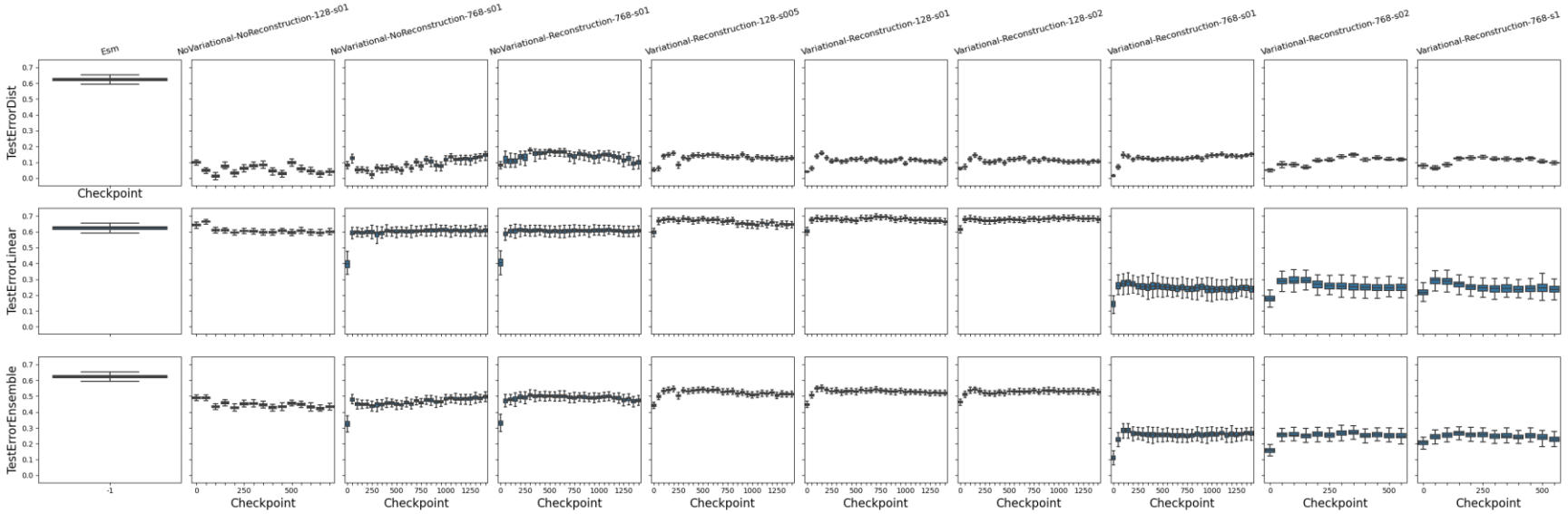


Figure 3-8. Results comparing downstream supervised processes using either SPICE embeddings or ESM embeddings when 1536 training examples are used from the AAV dataset. The y-axis is Spearman ρ and the x-axis is the number of training epochs. Results for nine variations on SPICE are presented. The only ESM result presented is from the six-layer pretrained model provided by the ESM repository. The same information as in Figure 3-3 – Figure 3-5 is presented here; only, rather than showing the " from-distribution," " linear," and " ensemble" predictions on the same plot, they are now each presented in a separate row. The first row of plots gives the results from-distribution predictions; the second row is for the linear model; the third row is the ensemble. Note that the plots in the first column are derived from a linear model applied to mean-pooled embeddings from the pretrained ESM model from the ESM GitHub repository—these results are only really comparable to SPICE results presented in the second row. Depending on the specific SPICE model variant tested, results from embeddings derived from ESM are comparable or superior to those derived from SPICE. Note again that, just like in Figure 3-3 – Figure 3-5, the from-distribution (first row) and ensemble (last row) calculations are only relevant to the variational SPICE models.

3.3.3 Active Learning with Gaussian Processes and SPICE

If applied in the laboratory, the predictive strategies discussed in the previous section would have required only two rounds of data collection: one to gather training data and another to evaluate predictions. Because every round of data collection will typically add multiple days (if not weeks) of laboratory time to an engineering endeavor, MLPE strategies like these that minimize the number of data collection rounds tend to be the most practically applicable. That said, there can be an advantage to adding more iterations of model training and evaluation to the engineering pipeline, particularly in situations where gathering experimental data is extremely expensive or the sequence space to be evaluated is otherwise large relative to the availability of labeled training data. Such approaches are typically referred to as "active learning," as the model used to guide exploration of sequence space is updated after each round of experimental data collection. When used in MLPE, the goal of active learning is to find some optimal protein in as few steps as possible.

In this section, I investigate the effectiveness of embeddings derived from SPICE and ESM for active learning, focusing in particular on their utility for Gaussian process-based navigation of protein sequence space. A Gaussian process (GP) is a distribution over functions defined by a multivariate normal distribution. At an intuitive level, the goal when training a GP is to identify the most probable distribution of functions that explains the observed training data; this can then be used to predict a distribution of possible values for previously unseen samples. Importantly, because predictions are returned as distributions, they provide an inherent sense of model confidence, with broader predicted distributions corresponding to less certain predictions and vice versa. Predictions made by GPs are thus extremely useful for active learning as they can be used to identify both (1) where the model expects optimal samples to reside and (2) where additional training data should be gathered to improve model predictive performance. In other words, at each iteration, GPs can be used to make an informed decision about what data to collect next.

To begin the learning process for a GP, an initial distribution of functions—a prior distribution—must be defined that informs the types of functions that can be learned; during

training, the distribution of functions defined by this prior is narrowed to one that is most probable given observed data. Importantly, this prior is also a GP and so is a multivariate normal distribution defined entirely by a mean and a covariance matrix. The mean defines the most likely function value at any given point prior to seeing any data while the covariance matrix defines the pairwise relationship between any *pair* of points, in turn defining the characteristics of the functions described by the GP. In practice, the mean is typically assumed to be zero and the covariance matrix is calculated from input data using what is known as a kernel function.[n] Simply put, a kernel function defines the degree of influence two points in the input space have on one another; this influence is then reflected in the calculated covariance matrix which, as a result, determines the shape of the functions described by the GP. While there are many options for kernel functions, most assume that points close to one another in the input space have the greatest influence on one another—that is, they assume that points close in space will have similar labels. Because of this assumption, GPs can be expected to perform best on explicitly informative embedding spaces where points with similar labels are expected to be near one another.

Because SPICE was designed to explicitly represent determinants of protein fitness in its embedding space, the embeddings produced from it might be expected to be particularly amenable to active learning with GPs. To test this hypothesis, I gathered several datasets spanning various protein activities[73,123,148,152,165–169] then simulated GP-based active learning on each dataset using either SPICE embeddings, mean-pooled ESM embeddings, or one-hot encoding to represent protein sequences (and so define the layout of the input space). For each of these active learning experiments, I started from the parent (unmutated) protein with its associated fitness value; using this single point, I trained a GP and used the resultant model

[n] Before training a GP, the first step is typically to center and scale the labels of the input training data such that their mean value is 0 and their standard deviation is 1. This sets up the problem such that the mean of the GP can be set to 0 as it is now, by definition, the expected (i.e., average) value across all points.

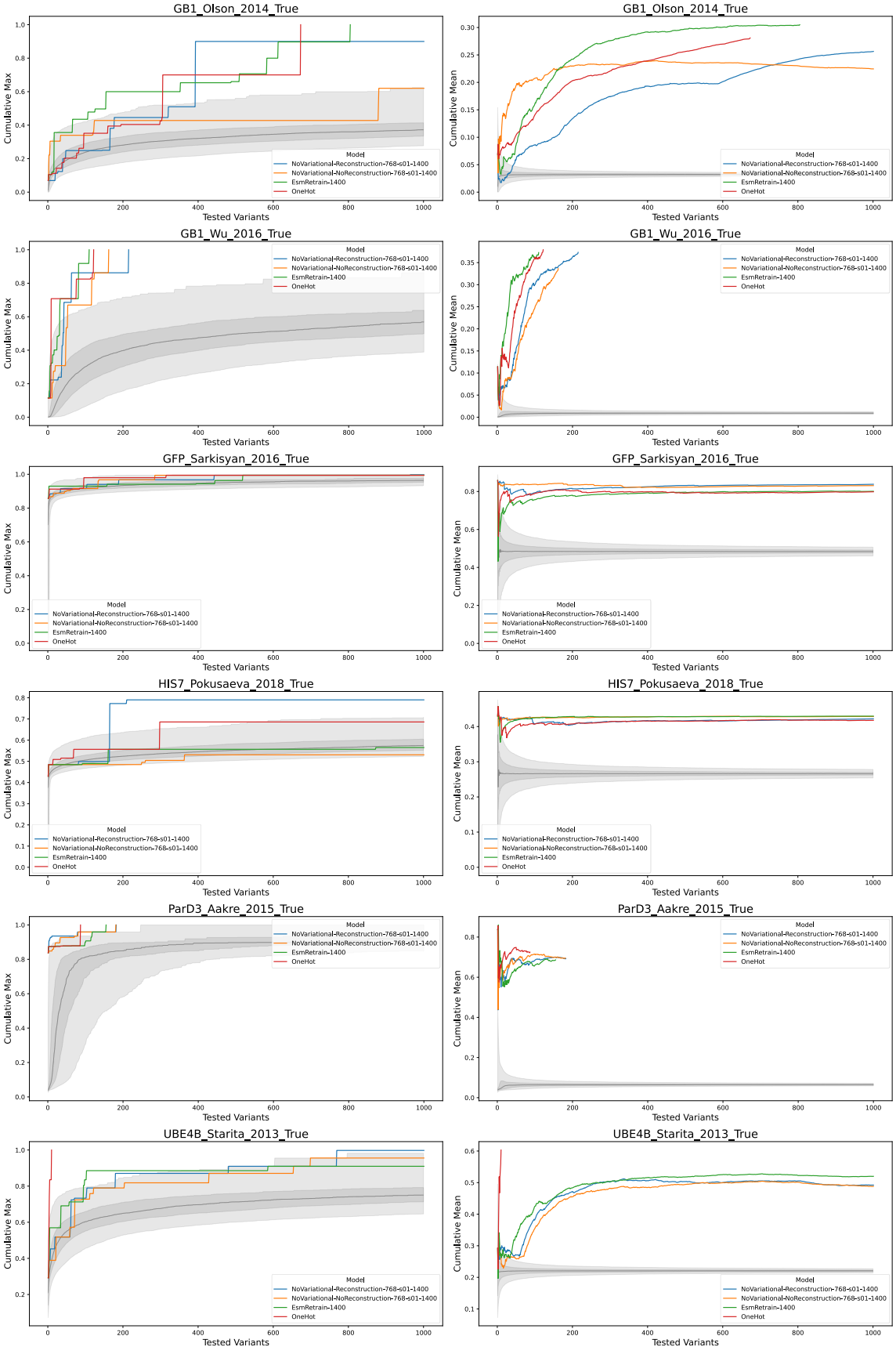
to select the next protein to evaluate using the upper-confidence bound (UCB) algorithm.[157,o] I iteratively retrained the model and used it to select the next point for 1000 rounds or until the protein variant with the highest fitness in the dataset had been identified. Each active learning experiment was repeated using eleven different kernel functions.

The results of the active learning experiments are presented below in Figure 3-9. Importantly, only the results from the best performing kernel for each dataset and encoding scheme are shown, where the best performing kernel is defined as the one that led to the collection of datapoints with the highest cumulative mean fitness upon completion of the experiment. While, in practice, it would be difficult to determine the best possible kernel for an encoding or dataset, here the goal is only to compare the *potential* of different encoding schemes for active learning with Gaussian processes. Thus, manually choosing the best-performing kernel is appropriate.

As in the previous section, the results here show no clear advantage of any encoding strategy over the others. Indeed, the best-performing encoding seems to depend strongly on the dataset tested. Strikingly, in many cases, one-hot encoding outperforms both semi-supervised approaches, again raising questions over the efficacy of existing unsupervised pretraining strategies. One-hot encoding will order proteins in the encoding space simply by their Hamming distance from one another (i.e., the number of mutations separating them). While there will certainly be a correlation between Hamming distance and protein fitness (generally speaking, proteins are increasingly less likely to share fitness as they are separated by more mutations), the strength of the correlation will depend on the location of mutations in the protein structure. For instance, single mutations to the surface of a protein will likely have minimal effect on fitness while single mutations to an active site can drastically alter it; the degree of correlation will thus be heavily context dependent. Extending this logic, it would

[o] The upper confidence bound (UCB) algorithm provides a principled way to balance exploration (i.e., gathering data from previously unexplored regions of the input space to improve model performance) and exploitation (i.e., gathering data from regions predicted to optimize the target objective) in GP-based active learning. The algorithm chooses the point that satisfies $\mathrm{argmax}_x\, \mu(x) + \beta\sigma(x)$ as the next to sample, where x is a candidate point, μ is the mean of the distribution predicted by the GP, σ is the standard deviation predicted by the GP, and β is a scaling factor for σ that sets the balance between exploration and exploitation. For the work in this chapter, I used $\beta = 1$ when employing the UCB algorithm.

be reasonable to assume that models like SPICE that explicitly capture this context in their embedding space would be more effective for GP-based active learning. Unfortunately, this is not what is seen here. Instead, these results show that both the SPICE and ESM embedding spaces are no more informative for GP-based active learning than one-hot encoding.


GB1_Olson_2014_True
Cumulative Max
Cumulative Mean
Tested Variants
GB1_Wu_2016_True
GFP_Sarkisyan_2016_True
HIS7_Pokusaeva_2018_True
ParD3_Aakre_2015_True
UBE4B_Starita_2013_True
Model
NoVariational-Reconstruction-768-s01-1400
NoVariational-NoReconstruction-768-s01-1400
EsmRetrain-1400
OneHot

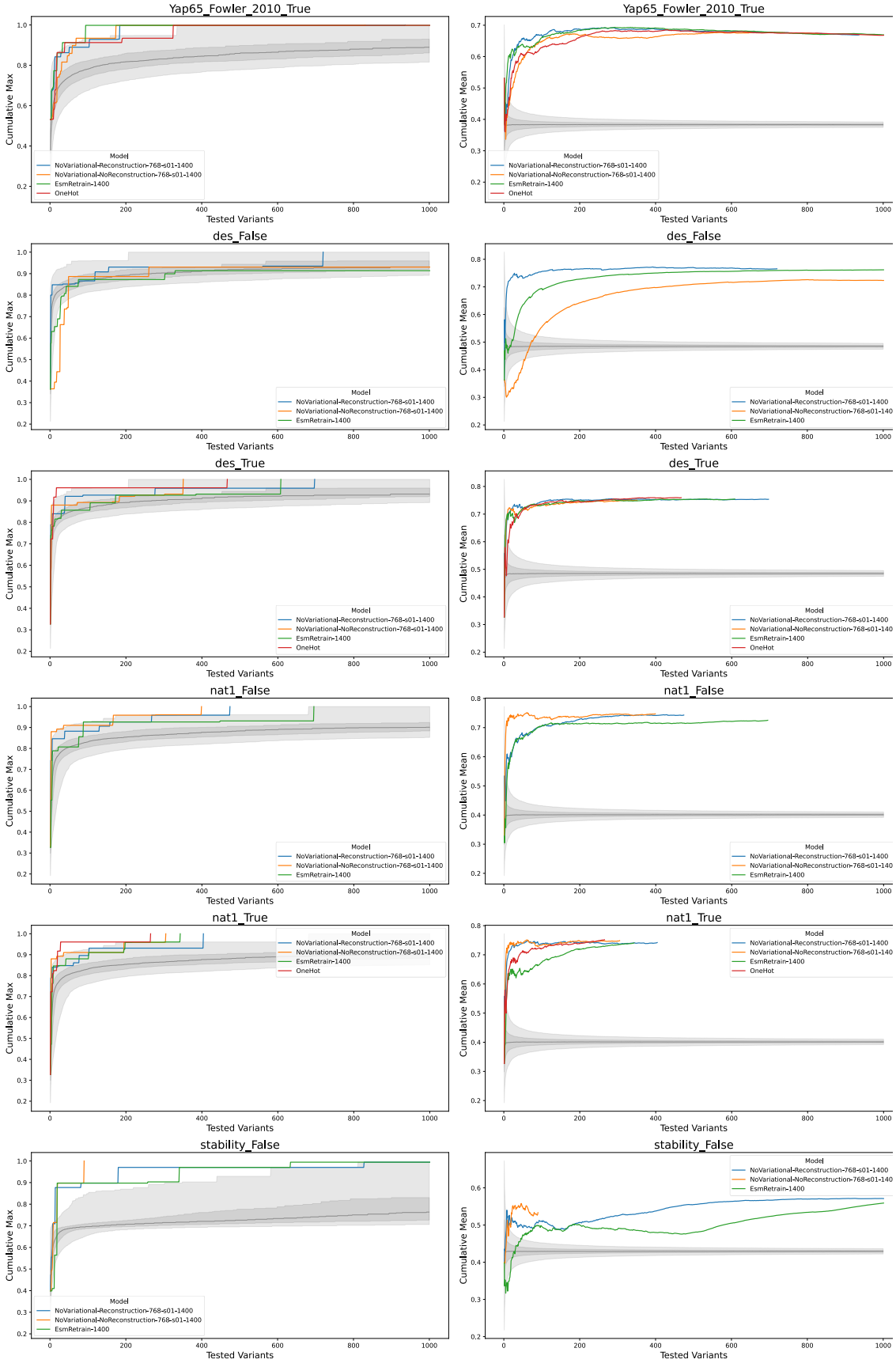

Yap65_Fowler_2010_True
Cumulative Max
Cumulative Mean
Tested Variants
Model
NoVariational-Reconstruction-768-s01-1400
NoVariational-NoReconstruction-768-s01-1400
EsmRetrain-1400
OneHot
des_False
des_True
nat1_False
nat1_True
stability_False

Figure 3-9. The cumulative max (left column) and mean (right column) achieved by the best possible kernel during 1000 rounds of active learning for four different encodings across nine different datasets. Orange and blue lines correspond to SPICE variants (both with and without the reconstruction loss added on), red lines correspond to one-hot encoding, and green lines correspond to encoding using ESM. The shaded grey area gives confidence intervals (2.5%, 25%, 50%, 75%, and 97.5%) for the same summary metrics had sampling been performed randomly. Each row in the plot corresponds to active learning results on a different dataset. In general, it is difficult to claim if any one embedding outperformed the others. Note that the last hyphen-delimited section of the SPICE names can be ignored—it only provides information about how long the model was trained during unsupervised pretraining.

3.4 Conclusions for Chapter 3

This project was designed to explore the potential utility of explicitly informative learned embeddings for semi-supervised protein fitness prediction. In general, I found that embeddings derived from my explicitly informative model—SPICE—provided no advantage over embeddings derived from existing state-of-the-art models across a number of different datasets and two different engineering strategies. While these conclusions were disappointing, the experiments performed to arrive at them also further reinforced an increasingly recognized trend: that existing unsupervised pretraining strategies build learned embedding spaces that offer little if any advantage over uninformative random or one-hot encodings for downstream supervised protein fitness prediction.[33,62,63,65]

In Section 3.3.2, I plotted the effectiveness of learned embeddings as a function of the level of pretraining, finding that increased unsupervised pretraining tended to have at best minimal, but often no, unpredictable, or even detrimental impact on the effectiveness of learned embeddings when applied to downstream supervised tasks. As far as I am aware, no one had published plots of this type before I performed this study; however, since then, another group has published similar results showing that, specifically for fitness prediction tasks, unsupervised pretraining appears to have limited if not a detrimental effect on the quality of learned embeddings.[161] The one exception to this trend in my work was the from-distribution predictions using variational SPICE embeddings on the stability benchmarking dataset, where increased pretraining led to a consistent improvement in downstream supervised performance. The determinants of stability should be explicitly represented in the

SPICE embedding space and the from-distribution prediction strategy was designed to take advantage of such organization. There may thus be value in further investigating strategies to develop both (1) explicitly informative embedding spaces and (2) downstream predictive strategies specifically designed to use them. That said, this conclusion is based off a single result, and the general ineffectiveness of SPICE embeddings when applied to other datasets in Section 3.3.2 and the active learning campaigns of Section 3.3.3 should perhaps temper such optimism.

It is important to note that the conclusions made in this chapter are with respect to semi-supervised fitness prediction only. The effectiveness of learned embeddings for applications such as structural prediction or remote homology detection appear to improve with additional pretraining, and there have been a number of successful applications of semi-supervised learning to these and related tasks.[92–98,161] Additionally, the unsupervised pretraining stage clearly extracts useful information from unlabeled data—zero-shot prediction (a strategy where predictions are made using models trained only on unlabeled data) using models like ESM would be impossible if this were not the case.[33,67,170] The major challenge in semi-supervised protein fitness prediction appears to remain, then, exactly how to best transfer the information learned during the unsupervised pretraining stage into the downstream supervised learning stage. Perhaps the most effective way to do this is to move away from embeddings entirely, instead using strategies like those proposed by Hsu et al. where zero-shot predictions made by unsupervised models are directly used to represent protein sequences.[63] That said, the effectiveness of learned embeddings in so many other biological tasks suggests that there should be a viable strategy for using them in fitness prediction—we just have to find it. Whatever the best strategy may be, for now it is clear that there is still much to be explored and learned about semi-supervised protein fitness prediction.

Chapter 4

EVSEQ: COST-EFFECTIVE AMPLICON SEQUENCING OF EVERY VARIANT IN A PROTEIN LIBRARY

Material from this chapter appears in **Wittmann, B. J.**; Johnston, K. E.; Almhjell, P. J.; and Arnold, F. H. (2022) evSeq: Cost-Effective Amplicon Sequencing of Every Variant in a Protein Library. *ACS Synth. Biol. 11*, 1313–1324. https://doi.org/10.1021/acssynbio.1c00592.

Abstract

Widespread availability of protein sequence-fitness data would revolutionize both our biochemical understanding of proteins and our ability to engineer them. Unfortunately, even though thousands of protein variants are generated and evaluated for fitness during a typical protein engineering campaign, most are never sequenced, leaving a wealth of potential sequence-fitness information untapped. Primarily, this is because sequencing is unnecessary for many protein engineering strategies; the added cost and effort of sequencing is thus unjustified. It also results from the fact that, even though many lower-cost sequencing strategies have been developed, they often require at least some sequencing or computational resources, both of which can be barriers to access. Here, I present every variant sequencing (evSeq), a method and collection of tools/standardized components for sequencing a variable region within every variant gene produced during a protein engineering campaign at a cost of cents per variant. evSeq was designed to democratize low-cost sequencing for protein engineers and, indeed, anyone interested in engineering biological systems. Execution of its wet-lab component is simple, requires no sequencing experience to perform, relies only on resources and services typically available to biology labs, and slots neatly into existing protein engineering workflows. Analysis of evSeq data is likewise made simple by its accompanying software, which can be run on a personal laptop and was designed to be accessible to users with no computational experience. Low-cost and easy to use, evSeq makes collection of extensive protein variant sequence-fitness data practical.

4.1 Introduction for Chapter 4

Engineered proteins are valuable tools across the biological and chemical sciences and have revolutionized industries ranging from food to fuels, pharmaceuticals, and textiles by providing green and efficient protein solutions to challenging chemical problems.[1] Over the course of a protein engineering campaign, hundreds to thousands or more protein variants will be constructed and have their fitnesses (level of, e.g., thermostability, catalytic activity, substrate binding, etc.) evaluated. Notably, sequence information is typically not gathered alongside the functional information, even though it could provide useful biochemical insight.[123,171,172] This is largely because many engineering strategies can be applied without sequencing. For example, during a typical directed evolution (DE) experiment, often only the best-performing variant or variants are sequenced in each round of mutagenesis and screening; sequencing every variant is viewed as an unnecessary expense. Given the massive amount of functional data gathered during a typical DE campaign, however, if sequencing *were* performed for the variants generated during these experiments, the resultant large datasets of sequence-fitness information could be revolutionary for biological, biochemical, and biocatalytic research. This is especially true for data-driven protein engineering strategies such as machine learning (ML), the development of which has benefitted tremendously from large sequence-fitness datasets made available by strategies like deep mutational scanning (DMS) and in databases like ProtaBank.[15,16,155,173,17,18,22,67,68,70,113,114]

Unfortunately, the standard sequencing strategy employed during DE—Sanger sequencing—is too expensive for sequencing all variants tested during a round of evolution.[81] Sanger sequencing is ubiquitous due to ease of sample preparation and ready availability of sequencing providers. However, the cost of Sanger sequencing scales linearly with the number of samples (Figure B-1). Thus, while the cost of sequencing just the top variants in a round of DE is minor, sequencing the hundreds or thousands of variants generated over the full engineering endeavor is not. Ideally, any new approach to sequencing during a protein engineering campaign would be comparable in cost, effort, and accessibility to that of sequencing just the top variants by Sanger sequencing. Here I present a collection of standardized and accessible protocols, components, and software that accomplishes this

goal. This collection, which I call every variant sequencing (evSeq), democratizes barcode sequencing strategies and expands on services made available by multiplexed next-generation sequencing (NGS) providers to allow amplicon sequencing of a region of interest within every variant produced during a round of DE at a cost of cents per variant.[83,174] Sample preparation for evSeq is simple, and the method requires no experience with NGS to perform, relies only on resources and services typically available to biology labs, and slots neatly into existing protein engineering experimental workflows. The accompanying software for analysis of evSeq data (found at github.com/fhalab/evSeq, documentation at fhalab.github.io/evSeq) was designed to be accessible to users with no computational experience and can be run on a personal laptop.

In this chapter, I detail the underlying strategies, protocol, and potential applications of evSeq. I begin by describing the strategies employed by evSeq to extend multiplexed NGS for sequencing protein variant libraries in a way that reduces both cost and effort. I then describe the wet-lab protocol of evSeq sample preparation, focusing on how it can be completed without disrupting an existing protein engineering workflow. Next, I discuss the features of the evSeq software before finally presenting two case studies that highlight potential applications of evSeq. In particular, I highlight how (1) the sequence-fitness data from evSeq can provide valuable information about the quality of variant libraries and the functional screen as well as how mutations modulate protein activity, and how (2) the data generated from evSeq can be used to implement ML for protein engineering. I designed evSeq for use as a routine procedure in many protein/enzyme assays (especially DE and protein engineering experiments leveraging mutagenesis strategies that target specific sites or a segment of the sequence). This tool brings cost-effective, easy-to-use sequencing to all protein engineers, regardless of experience with NGS and access to sequencing and computational resources. I believe that widespread adoption of evSeq—and the resultant datasets generated—will be invaluable for future ML-guided protein engineering and will help us better understand protein sequence-fitness relationships.

4.2 Results and Discussion for Chapter 4

4.2.1 evSeq Uses Inline Barcoding to Expand on Commercially Available Multiplexed Next-Generation Sequencing

Unlike Sanger sequencing, which outputs a single chromatogram that represents the population of DNA in a sequenced sample, NGS outputs millions of individual DNA reads that represent a random draw from the population of DNA in the sequenced sample.[83] Confidence in NGS sequencing results is largely determined by the sequencing "coverage," which for the purposes of this chapter is defined as the number of returned reads that map to a specific nucleotide on a reference sequence. Higher coverage enables more confident identification of mutations relative to a reference sequence as the increased redundancy allows distinguishing between true sequence mutations and errors that arise during library preparation, clustering, or sequencing.

A single NGS run is roughly three orders of magnitude more expensive than a Sanger sequencing run, but because the run outputs millions of reads, this cost can be spread over multiple samples using a technique known as "multiplexed NGS" (Figure B-1). In multiplexed NGS, each submitted sample is tagged with a "molecular barcode"—a unique piece of DNA that encodes the sample's original identity—before all samples are sequenced together in the same NGS run.[174–180] Post sequencing, the barcodes are used to assign individual reads to individual samples. For instance, barcodes can be used to distinguish reads coming from samples belonging to specific plates and wells.[181] Importantly, multiplexed NGS can be outsourced just like Sanger sequencing (making it accessible to all laboratories regardless of sequencing experience), and sequencing providers typically charge tens of dollars per sample in a multiplexed sequencing run, yielding on the order of 10^4–10^5 individual sequences per sample (assuming the run is performed on an Illumina MiSeq instrument).

The level of coverage granted by a set number of reads depends on the length of the DNA sample being sequenced, the length of the NGS read used to sequence it, and whether those reads are paired-end. NGS reads are short (300 bp or less on Illumina systems), and so reads

must be spread across a longer sample to sequence it in full. The expected coverage (average coverage per nucleotide) obtained for a DNA sample thus depends both on its length and the read length used for sequencing. For example, with the ~10^5 reads returned by a commercial MiSeq multiplexed sequencing run, a 3 Mb genome could be sequenced with 150 bp paired-end reads to an expected coverage of ~10x, whereas a 20 kb plasmid could be sequenced to an expected coverage of ~1500x.

Because shorter samples can be sequenced at higher coverage for a given number of reads, it can be advantageous to sequence only the region of interest of a sample. This is exemplified by amplicon sequencing, a strategy in which a researcher sequences a PCR product (an amplicon) that targets a specific region of interest in the DNA.[182] For instance, continuing the example from above, with ~10^5 total 150 bp paired-end reads, a 300 bp PCR product could be sequenced to an expected coverage of ~100,000x.

Many mutagenesis methods employed in protein engineering (e.g., site-saturation[183] and tile-based mutagenesis[184] strategies) target mutations to a specific position or region in the sequence of a protein, and thus the variants produced can be sequenced with amplicon sequencing to high coverage.[175] Notably, however, even though increasing coverage yields more confident results, it comes with diminishing returns, and it is generally held that coverage in the tens is more than sufficient for effective reference-based identification of mutations (Figure B-1).[185] Indeed, clinical sequencing of human genomes targets 30x coverage or greater to minimize false base calls. Given this reference, it is clear that the ~100,000x coverage that would be returned from a multiplexed sequencing run for a 300 bp amplicon is far higher than necessary for effective identification of mutations—2,000 amplicons could be sequenced in the same run and still yield clinical-grade coverage.

evSeq achieves cost-effectiveness by relying on the facts that (1) at tens of dollars per sample, the cost of sending a single sample to an outsourced multiplexed NGS run is comparable to the total cost of Sanger sequencing the top variants in a round of DE, (2) amplicon sequencing can be used to identify mutations in protein variants from many protein engineering library types, and (3) enough reads are returned for a single sample in a commercial multiplexed

NGS run to sequence hundreds of amplicons. Specifically, the evSeq protocol (Figure 4-1, B.2.3: evSeq Library Preparation/Data Analysis Protocol) works by focusing all reads of a single multiplexed NGS sample to specific regions on hundreds of protein variants, achieving sequencing depths of 10^1–10^3 at the approximate cost and level of accessibility of using Sanger sequencing of just the top variants in a round of DE (Figure B-1).

The evSeq library preparation protocol begins with PCR amplification of the region of interest in each variant (i.e., the position/region where mutations were made) and appending inline DNA barcodes to the resultant amplicons that encode their original plate-well position (Figure 4-1A).[181,186,187] This is a one-pot, two-step, plate-based PCR procedure that uses two sets of primer pairs. Each primer in the first set of primers ("inner" primers) consists of a user-specified 3' "seed" region that binds to the regions flanking the region of interest as well as a 5' predefined universal adapter (B.1.6.1: Inner Primer Design). Each primer in the second set of primers ("outer" primers) consists of (1) a 3' region that matches the adapter of the inner primers, (2) a central 7-nucleotide barcode where each barcode pair between forward and reverse outer primers is unique to a plate-well position, and (3) a 5' sequence matching the Illumina Nextera transposase adapters (B.1.6.2: Outer Primer Design, B.1.6.3: Barcode Design, Table B-1, Table B-2). I designed 96 unique forward and 96 unique reverse outer primers for evSeq which, because both forward and reverse outer primers contain a barcode, can be combined to encode up to $96^2 = 9{,}216$ possible plate-well positions (B.2.2: Preparation of evSeq Barcode Primer Mixes, Table B-3 – Table B-10. Note that a pre-filled IDT order form is also provided for the outer primers on the GitHub associated with this work—see B.2.1: Ordering Barcode Primers from IDT for details. While it is recommended to use these pre-tested barcodes, users can also design their own to, e.g., further expand the number of available combinations.). Importantly, this set of outer primers can be used to sequence any target region from any gene with evSeq, and so only needs to be ordered once, constituting a one-time initial setup cost in the range of around a thousand dollars (the exact cost will vary based on oligo provider and any institutional agreements set up with said provider). Once outer primers are ordered, only a new inner primer pair is needed for each new region of interest to be targeted by evSeq.

Once all barcoded amplicons have been produced, they are pooled and sent to a sequencing provider, who will then use the transposase adapters installed with the outer primers as a handle to perform a third and final PCR to barcode the *pool* of amplicons *once again* with a pair of sample-specific Illumina indices (Figure 4-1B). At this point each amplicon in the pool has one pair of sample-specific Illumina barcodes and one pair of plate-well-specific inline evSeq barcodes. This complete evSeq library is sequenced as a single sample in a multiplexed NGS run along with samples from other users (whether or not they are also evSeq samples). Post sequencing, the sequencing provider uses the sample-specific barcodes to identify those sequences belonging to the evSeq pool and returns them to the user (i.e., the provider "demultiplexes" the run, separating evSeq sequences from those of other users). The user then uses the evSeq software to analyze the returned sequences, assigning them to corresponding plate-well positions using the evSeq barcodes and identifying the mutations in the variants relative to a reference (Figure 4-1B, Figure 4-1C).

4.2.2 evSeq Library Preparation Fits Into Existing Protein Engineering and Sequencing Workflows and Was Designed to be Resource Efficient

A typical procedure for evaluating protein variants involves (1) arraying colonies of an organism (e.g., *Escherichia coli*) that harbor a plasmid encoding a protein variant into the wells of a (usually 96-well) microplate, (2) growing the resulting cultures to stationary phase (colloquially, an "overnight culture"), (3) using the overnight culture to inoculate a fresh culture that will be used to express the protein variants, and (4) evaluating the fitnesses of expressed protein variants. The expression stage (step 3) typically involves downtime where the experimentalist must wait until the culture reaches sufficient density before inducing protein expression and then again as expression takes place. evSeq library preparation can be performed easily in either of these time windows. The evSeq library preparation protocol begins with the barcoding PCR described at the end of the previous section; this one-pot, two-step, plate-based PCR was designed to be compatible with outsourced sequencing workflows, minimize preparation time, and minimize laboratory resource usage (B.2.3: evSeq Library Preparation/Data Analysis Protocol). For instance, use of inline barcodes is a known, effective strategy for expanding the number of samples that can be multiplexed

without having to modify the Illumina indices used during multiplexed sequencing.[186,187] Because evSeq library preparation uses inline barcodes, it grants the outsourced sequencing provider maximal flexibility in choice of Illumina indices. In other words, evSeq library preparation is decoupled from preparation of the Illumina library that will eventually be sequenced, allowing the evSeq library to be run just as any other sample would be that is submitted to a sequencing provider.

As mentioned in the previous section, use of a two-step PCR reduces the number of primers that must be ordered per new sequencing region of interest. Because evSeq relies on 96 unique forward barcodes and 96 unique reverse barcodes, a single-primer PCR would require ordering 192 new barcoding primers for each new target region evaluated in each library. In a two-primer protocol, however, the inclusion of a universal adapter on the inner primers allows the same 192 outer primers to be used regardless of target position in the variant—only two unique primers (forward and reverse inner) must be purchased for each new target region, and only if existing inner primers from previously targeted regions are not already compatible. Additionally, the evSeq PCR directly uses liquid from the overnight culture as a source of template DNA (Figure 4-1B, B.2.3: evSeq Library Preparation/Data Analysis Protocol); the template DNA is released from lysed cells during the initial heating step of the PCR, avoiding a costly and time-intensive DNA isolation/purification step and allowing researchers to use materials already prepared as part of the protein expression workflow.[187]

The remaining steps of evSeq library preparation were, like the PCR stage, also designed to be resource and time efficient. After completion of the PCR, the resulting barcoded amplicons are pooled by plate and purified via gel extraction. Pooling prior to purification goes against standard practice for multiplexed NGS library preparation, which is to purify samples individually, quantify their DNA concentration, then combine them in equimolar quantities to ensure more equal read distribution across samples after sequencing.[188] However, because individual plates in protein engineering libraries tend to contain variants from the same region of the same protein scaffold (e.g., as would be typical for variants from a comprehensive site-saturation library), it is assumed that the variation in PCR reaction yield

will be minor within plates and that, as a result, the same plate can be pooled prior to quantification with only minor effects on read distribution. Using this "pooling first" strategy, only as many purifications as there are *plates* must be performed as opposed to as many as there are *variants*, thus enabling faster processing of evSeq amplicons while reducing resource usage. As will be shown in later sections, the distribution of reads returned using pooling first is perfectly acceptable for confidently identifying variant sequences.

Once all pooled plates have been purified, the concentrations of the individual purified pools are measured. The pools are then normalized by molarity and combined into a final evSeq library, which is in turn submitted as a single sample to a sequencing provider. As described in the previous section, the provider will perform a final PCR on the evSeq library to add sample-specific barcodes before sequencing it as a single sample in a multiplexed sequencing run. Outsourcing the sequencing stage has two main benefits: First, it allows evSeq to be performed by research groups with no prior sequencing experience and no direct access to sequencing equipment—groups need only be familiar with PCR, a ubiquitous technology in protein engineering laboratories. Second, to be cost effective, multiplexed sequencing should be run with tens of samples at least (Figure B-1). By outsourcing the sequencing stage, groups that do not frequently produce evSeq libraries need not wait until enough libraries have accumulated to run sequencing—a single outsourced submission, for instance, can be run along with those of other research groups with a variety of different sequencing needs.

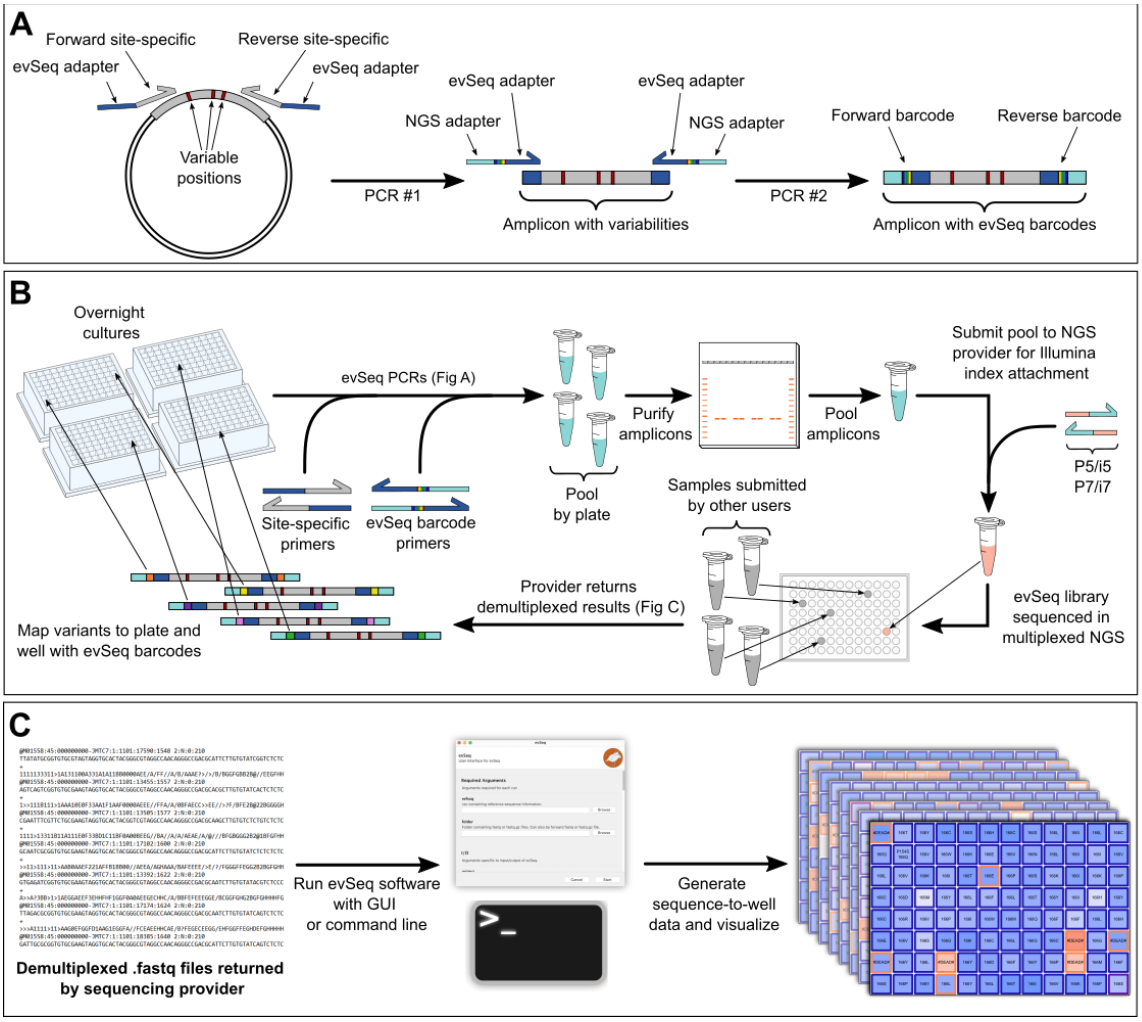


Figure 4-1. Overview of evSeq library preparation and processing. (A) In the first stage of the PCR, a region of interest is amplified with primers that include a 3' site-specific region (gray) with 5' adapter sequences (dark blue). The second PCR stage adds molecular barcodes (rainbow) with primers that bind to the adapter regions and add adapters for downstream NGS processing (light blue). (B) To avoid costly DNA isolation steps, evSeq uses liquid cultures of cells harboring mutated DNA (e.g., an "overnight culture" of E. coli) as template during the one-pot two-step barcoding PCR described in A. Each plate is pooled individually and gel purified. Purified pools are then adjusted for concentration differences and pooled together before being sent to a sequencing provider, who then appends another set of barcodes as well as sequence elements necessary for Illumina NGS sequencing. This sample is now pooled with those of other users and a multiplexed sequencing run is performed. After sequencing, the sequencing provider uses the barcodes that they attached to separate ("demultiplex") the evSeq reads from reads of other users; the provider returns evSeq reads in .fastq files. (C) The .fastq files returned by the NGS provider are inputs to the evSeq software, which uses the evSeq forward/reverse barcode pair to map each read to a specific plate and well based on known barcode combinations. The software also processes the

mapped reads (see Appendix B and evSeq documentation for more details) to, among other things, assign variant identities to each well and return interactive HTML visualizations.

The final stage of the evSeq workflow is data analysis using the evSeq software (github.com/fhalab/evSeq) (Figure 4-1C). Extensive documentation of the software and its capabilities is available as a website (fhalab.github.io/evSeq). The software was designed to be accessible to users with varying degrees of computational experience and can be run through either a graphical user interface (GUI), a command line application, or in a Python environment (e.g., a Jupyter notebook). Outputs from the software range from high-level overviews of data (e.g., an interactive "Platemap" graphic that displays sequencing coverage and identified mutations in each well of each plate; see Figure 4-1C for an example) to low-level details about the population of reads assigned to each well (e.g., in a well identified as polyclonal, the percentage of reads mapping to each of the identified variants). Functional data can also be easily associated with identified variants using the evSeq software outputs to produce sequence-fitness datasets, and Jupyter notebooks and web pages are provided that walk users through the process.

4.2.3 evSeq Facilitates Library Construction, Validation, and Sequence-Fitness Pairing

To highlight the utility of evSeq for engineering and biochemical experiments, my coworkers and I first examined how it could be used to construct high-confidence and informative sequence-fitness data. Specifically, my coworkers constructed and screened eight single-site-saturation libraries of the enzyme Tm9D8*—an engineered β-subunit of tryptophan synthase from *Thermotoga maritima* (*Tm*TrpB)—for tryptophan-forming activity at 30 °C (Figure 4-2).[189] In two of the screened libraries, they targeted two positions distant from the active site (A118 and S292) that have been seen to play a role in allosteric regulation of *Tm*TrpB enzymes; in the other six libraries, they targeted active-site residues known to modulate the activity of TrpB (E105, L162, I166, F184, S228, and Y301) (Figure 4-2A).[101,190,191] As shown below, this type of sequence-fitness data can be used to assess the quality of a protein

engineering library, identify improved variants during a round of directed evolution, and give insight into the significance of a given residue in catalysis.

Many factors can introduce bias into a site-saturation mutagenesis experiment, such as annealing bias for the native nucleotides during the PCR for library construction or contamination with the template plasmid during transformation. Without sequencing all of the variants, it is impossible to know that the library is representative of the experimental design. Since evSeq reports exactly which variants are contained in a library, researchers can leverage this to implement important quality control practices as part of the standard protein screening workflow. For instance, of all 153 possible unique variants in the eight single-site-saturation libraries, we observed 149 of them (Figure 4-2B and Figure 4-2C); only I166A, S292C, S292D, and S292H could not be assigned with confidence, where we defined ≥80% abundance in a well with ≥10 reads as our confidence threshold. Of the variants identified, many were found in replicate (Figure 4-2D) due to oversampling during colony picking, which ensures that all protein variants have a chance to be found and screened (All libraries were constructed with the 22-codon trick[85] and 88 individual colonies were screened for each library, so we expected a 98% probability of seeing all variants assuming perfect construction of libraries). Conveniently, this oversampling also allowed us to evaluate the noise in our functional screen (Figure 4-2E), which further improved the confidence in the quality of data gathered.

Given just the fitness data gathered in this experiment, a protein engineer would identify 50 wells that are at least 1.2-fold improved over the parent enzyme Tm9D8*. However, with the sequence-fitness pairs constructed via evSeq, we know that these 50 *wells* correspond to only 16 unique *variants*. Depending on how conservative the engineer was as to what should be sequenced, a decision to sequence hits with Sanger sequencing could result in anywhere from 12 (2-fold improvement) to 50 (1.2-fold improvement) wells sent off for sequencing for a total cost of $36 to $150 (using an estimate of $3 per sequence). It would cost ~$2000 to sequence all eight plates via Sanger. Using evSeq, however, we obtained the sequences of *all* 625 wells of variants for only $100, corresponding to $0.13 per non-control well. In other

words, using evSeq, we can produce far more sequence-fitness information than sequencing just the top hits using Sanger all for a similar cost. Importantly, although the evSeq defaults currently allow only eight plates to be sequenced at once, the number of variants included in this experiment could likely have been increased as the median number of reads per well was 86 (mean: 98), which is above what is needed for reliable sequencing. Assuming that doubling the number of plates would halve the number of reads seen for each well, doubling the number of plates sequenced would cause only 14 non-control well sequences to drop below the confidence threshold.

The per-variant cost of evSeq may be reduced even further using different services and sequencing platforms. For instance, in both this section and the next, the reported number of reads and ~$100 total cost are from outsourced MiSeq runs, which returned hundreds of thousands of total reads per evSeq library. These numbers are reported because outsourced multiplexed MiSeq is a standard service available to all research groups. As an alternative to outsourcing, however, Caltech provides multiplexed sequencing (via the Caltech Millard and Muriel Jacobs Genetics and Genomics Laboratory) on an Illumina NextSeq platform, returning an average of ~10x more reads than the outsourced MiSeq run for a total cost of ~$10. At 10x more reads and 10x less the total cost, the per-variant evSeq cost could decrease 100-fold to <$0.01. Indeed, we were able to re-sequence the TrpB libraries at a per-variant cost of ~$0.01 with ~2.2 million total reads returned for an average of thousands of reads per variant, far higher than what is needed for reliable variant calling. It must be noted, however, that analysis of the millions of evSeq reads was no longer practical on a personal laptop, requiring a desktop workstation instead. Computational power beyond a laptop will be needed when processing more than hundreds of thousands of reads with the existing evSeq software.

Of final note, aside from providing valuable information for protein engineering experiments, evSeq can also facilitate investigation into the biochemical relevance of specific positions/mutations. Specifically, because all possible variants in a site-saturation library can be identified by evSeq, the sequence-fitness information generated can be used to explore

the effects of mutations more fully than, for instance, an alanine scanning experiment.[192] Using an example from the TrpB data gathered here, an alanine scanning experiment would tell a biochemist that the mutation to the conserved catalytic residue E105A inactivates the enzyme, with no information about the effects of other amino acid changes at this position. Using site-saturation with evSeq, we instead find that all mutations to E105 except for E105D inactivate the enzyme. The fact that glutamate and aspartate are the only amino acids containing a carboxylic acid suggests that this functional group is critical for activity (Figure 4-2E, with inset).

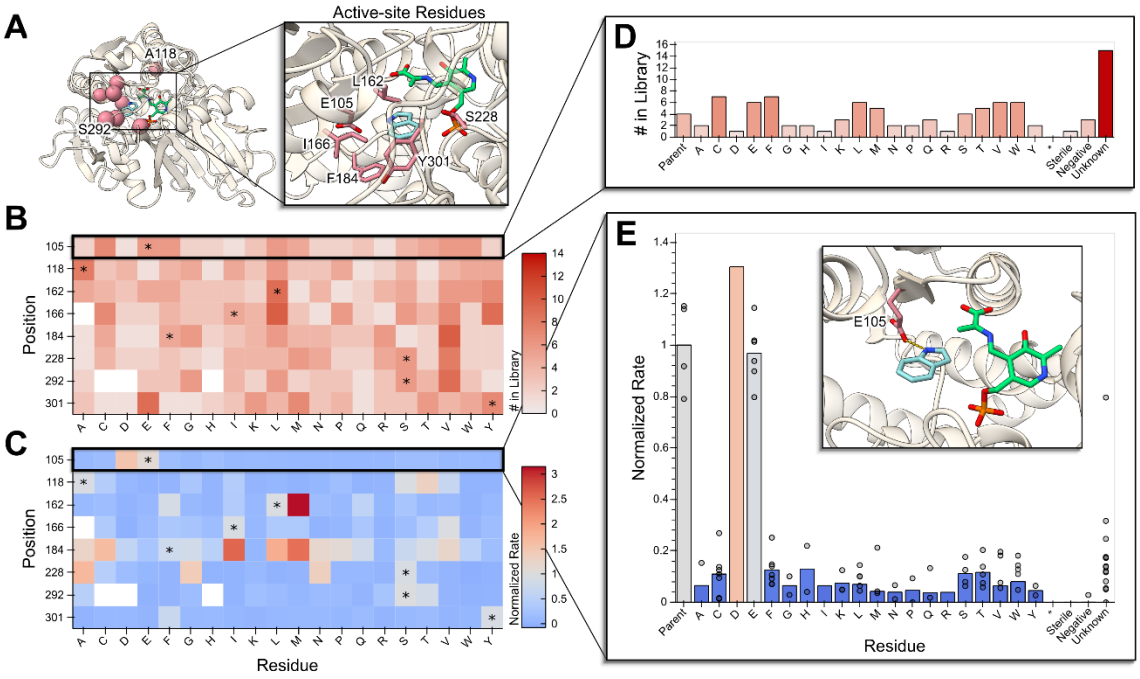


Figure 4-2. evSeq enables low-cost investigation of library quality and sequence-fitness pairing in site-saturation mutagenesis libraries. (A) Eight residues (red) known to modulate the activity of Tm9D8* were independently targeted with site-saturation mutagenesis: A118 and S292 (distal residues), E105, L162, I166, F184, S228, and Y301 (active-site residues). An active form of the pyridoxal 5'-phosphate cofactor is represented in green, and the substrate indole is shown in light blue. (B) Library quality can be investigated by plotting a heatmap of the number of times each variant/mutant was identified at each targeted position ("Counts") from processed evSeq data. Parent amino acids are each marked with an asterisk. (C) Likewise, the effect of mutations and mutational "hotspots" can be identified by plotting a heatmap of the average activity measurements for each variant/mutation in each library, normalized to the average parent activity for that library ("Normalized Rate"), when fitness data is combined with evSeq data. (D) An example plot made possible by evSeq visualization functions shows the number of times each amino acid was found in a single TrpB library (position 105), also accounting for known controls and unidentified wells. (E) Another example output of the evSeq software shows activity for a single library (position 105), showing biological replicates. The inset displays the role of the mutated residue in this library, which is to coordinate the nitrogen of the indole substrate. Note that the circles in this plot correspond to individual measurements while the bar plot represents the mean of these measurements. If no circles are present for a bar (e.g., E105D), then this is because only a single instance of this mutation was observed. Circles are not shown in this case to allow distinguishing between a single replicate and a tight distribution of multiple replicates.

4.2.4 evSeq Can be Used to Generate Data for Machine Learning-Assisted Protein Engineering

I next wanted to demonstrate the utility of evSeq for advancing and applying machine learning-assisted protein engineering (MLPE). In MLPE, models are trained to learn a function that relates protein sequence to protein fitness (i.e., they learn f(sequence) = fitness).[15–18,22] These models are then used for rapid, low-cost *in silico* prediction of protein fitness, avoiding or greatly reducing the need for often-costly laboratory screening of variants (Figure 4-3).

Sequence-fitness data is critical for effective MLPE. Indeed, even though strategies exist that *can* predict protein fitness from sequence alone (e.g., those that use evolutionary data to predict protein fitness), their effectiveness is improved with the inclusion of sequence-fitness information.[63,67,68,70] As a result, the most effective MLPE workflows require that both sequence *and* fitness data be collected, unlike a DE workflow, which requires only fitness data.

The need to collect sequence data in addition to fitness data is an often-overlooked additional cost of MLPE strategies compared to standard DE. Take, for instance, the machine learning-assisted directed evolution (MLDE) strategy described in Chapter 2.[33,82] In my first project at Caltech, I worked with others to use MLDE to evolve *Rhodothermus marinus* nitric oxide dioxygenase (*Rma*NOD) for greater enantioselectivity in a carbon–silicon bond-forming reaction.[82] Over the course of the engineering campaign, we collected six 96-well plates of sequence-fitness data for training ML models. In total, sequencing the variants in these plates by Sanger sequencing cost ~$1700. High additional sequencing costs like these can make MLPE methods far less attractive, even if they are more effective than traditional DE at finding high-fitness protein variants.[33] However, given that evSeq enables sequencing all variants for a cost similar to standard DE methods, it enables use of MLPE without added cost. In essence, evSeq eliminates the sequencing burden of MLPE.

To demonstrate the application of evSeq to MLPE, I used it to sequence five plates of *Rma*NOD variants from a four-site combinatorial library. Coupled with fitness data, the

sequences resulting from this run could be used to drive a round of MLDE. Notably, sequencing these plates by Sanger sequencing would have cost ~$1400; in contrast, sequencing by evSeq using an outsourced multiplexed MiSeq run cost ~$100 for a per-variant cost of ~$0.21. The median read depth per variant in this run was 463 (mean: 506), much higher than is required for accurate sequencing, and so more plates—from either the same or a different library—could have reasonably been added to this evSeq run to decrease the per-variant sequencing cost even further. Of course, as discussed in the previous section, in-house sequencing could have cut sequencing costs an additional ten-fold.

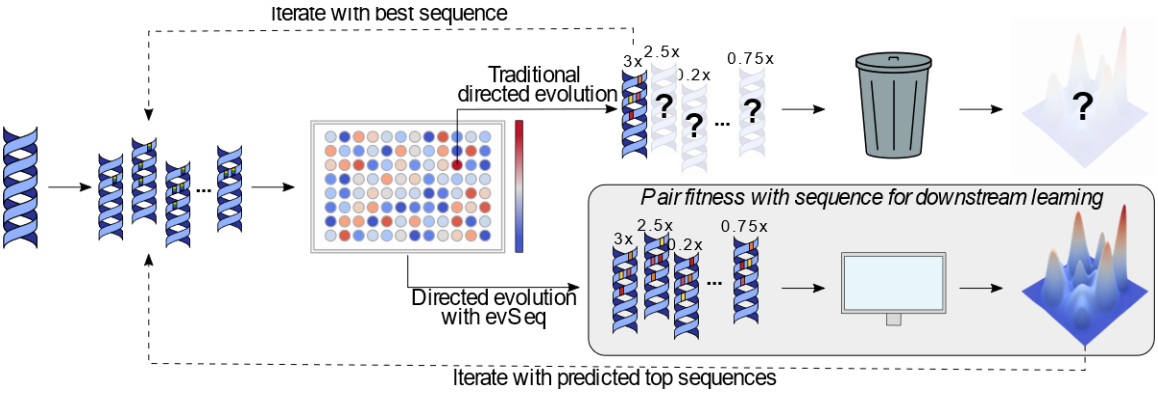


Figure 4-3. evSeq eliminates the sequencing burden of MLPE. Traditional DE only collects sequence information for top variants, essentially "throwing away" fitness data from inferior variants and learning nothing about the underlying fitness landscape. If, instead, evSeq is used to collect sequence information for all variants, MLPE methods, which require sequence-fitness pairs for supervised model training, can be implemented. Sampling from a fitness landscape, an ML model can be trained to predict the fitnesses of missing sequences and reconstruct the missing regions of this landscape.

The cost of sequencing is most notably a barrier for MLPE strategies that focus on developing models for a single protein with a well-defined fitness (e.g., MLDE); however, the applicability of evSeq to MLPE is not limited solely to cost-reduction. For instance, ML strategies have been developed that, rather than focusing on a specific protein, train models on sequence-fitness information across multiple different protein scaffolds.[173,193] The goal is for these models to learn global determinants of protein fitness, then to use the models as general-purpose protein fitness predictors. By enabling the collection of sequence-fitness pairs across a wider array of proteins and fitness definitions, evSeq opens these approaches

to new and more diverse data sources. Generally speaking, the more sequence-fitness data available to train and benchmark these strategies, the better we expect them to perform and the more rapidly we expect improvements to be developed.[173] It is no coincidence that large leaps forward in other ML disciplines have followed increased availability of large, diverse datasets, with the rapid advance in computer vision sparked by ImageNet being perhaps the most prominent example.[109] Widespread adoption of evSeq—and commitment to depositing sequence-fitness data in resources such as ProtaBank—would provide such a dataset for protein engineering.[113] This dataset would span the range of all engineered proteins and all target fitnesses, capture examples of sequences with both higher and lower/zero fitness relative to a parent (the latter of which is effectively never recorded with current DE sequencing practices), and overall enable rapid advancement in MLPE.

4.2.5 evSeq Detects All Variability in the Sequenced Amplicons

Although in this chapter I focused on demonstrating applications involving targeted mutagenesis strategies, evSeq is also applicable to other mutagenesis methods as the associated software can identify both user-specified and unspecified positions of variability (Figure 4-4A). This feature not only informs the user of potential unexpected mutations in the sequenced amplicon (Table B-11), but also allows it to work effectively with tile-based mutagenesis strategies and other semi-targeted mutagenesis strategies (e.g., error-prone PCR of specific regions or small genes). All that is required is that the amplicon length and read length be able to capture the full region containing mutations.

It should be noted that evSeq will not detect off-target mutations outside of the constructed amplicon as these regions are not sequenced, meaning that it is unable to identify other mutations in a larger DNA element that may be contributing to activity. Due to this fact, for exceedingly unexpected mutational effects that are not seen in replicate, it is suggested to sequence the rest of the DNA element to confirm the presence or absence of any off-target mutations. However, this limitation is mitigated by the fact that off-target mutations are rare and, importantly, evSeq is agnostic to read length and will work with any length of paired-end sequencing.[194]

While the current software version is not yet suited for other, long-read sequencing technologies (e.g., PacBio or Oxford Nanopore), future versions could be updated and validated with these data formats and make full gene-length evSeq experiments more straightforward and cost effective. Given this, evSeq is currently best suited and most cost effective when all expected mutations exist in the sequenced amplicon, though sequencing of multiple overlapping amplicons can readily allow evSeq to be expanded to sequence entire genes of variants arrayed in microplates (Figure 4-4B). Care must be taken in such an application, however, to account for the fact that aggressive mutation rates could compromise the annealing efficiency of inner primers binding in the variable region, as could mutations to positions closer to the binding region of the 3' end of the inner primer. Such situations would lead to a higher proportion of wells failing sequencing.

4.3 Conclusion for Chapter 4

Hundreds to thousands of protein variants (or more) are constructed and their fitnesses evaluated over the course of a standard protein engineering campaign. Without sequencing, these fitnesses are next to useless—the time, effort, and resources expended to produce them are largely wasted. Comparable in cost to existing protocols, accessible to scientists with no or minimal sequencing and computational experience, and easy to implement with existing technology, evSeq rescues these fitness data by making the collection of sequence data for every variant a practical and highly useful step of the protein engineering pipeline. Given the number of research groups working on DE and other protein engineering projects, widespread adoption of evSeq would lead to an explosion in the availability of sequence-fitness information. By sequencing every variant, no laboratory screening effort is wasted, and we open the door to advances in both our biochemical understanding of proteins and our ability to engineer them with data-driven methods.

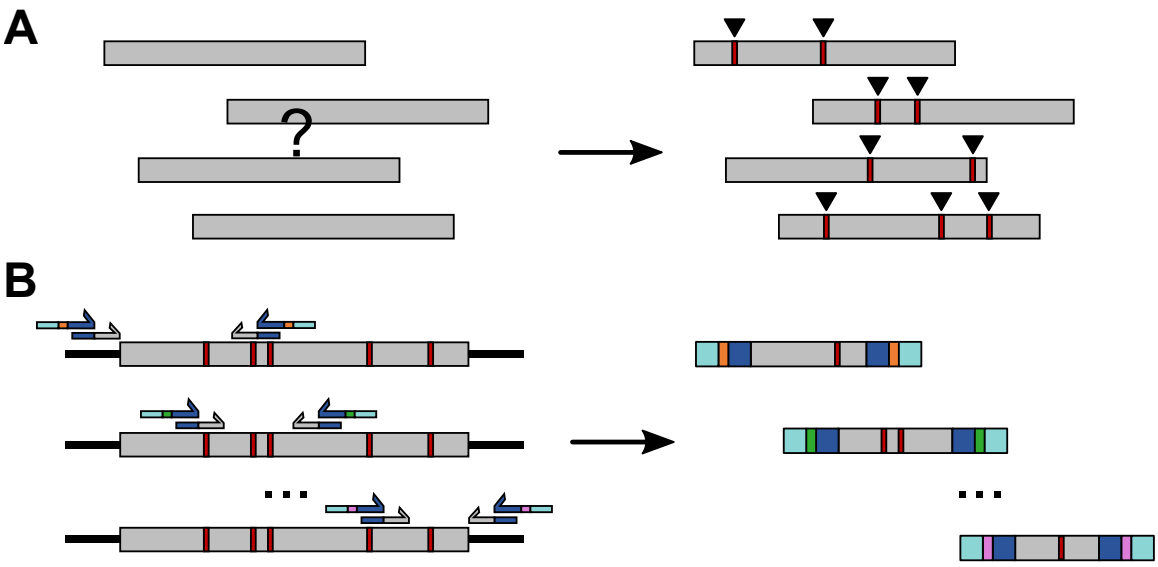


Figure 4-4. evSeq detects variability and can be expanded for random mutagenesis. (A) evSeq does not require that the user specify which position in the amplicon was targeted. Instead, the software can identify variable regions by comparing to a reference. (B) evSeq can be used to sequence entire genes by designing a set of inner primer pairs which together capture the entire gene. Different evSeq barcodes can then be used for each region, and the user can reconstruct the entire sequence.

4.4 Financial Support for Chapter 4

This work was supported by an Amgen Chem-Bio-Engineering Award (CBEA). This work was supported by the NSF Division of Chemical, Bioengineering, Environmental and Transport Systems (CBET 1937902). This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Award Number DE-SC0022218. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States

Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Appendix A

SUPPORTING MATERIAL FOR CHAPTER 2

A.1 Data and Code Availability

Data needed to replicate simulations have been deposited at Caltech Data and are publicly available. DOIs are listed in Table A-11. The raw simulation data reported in this study cannot be deposited in a public repository because it is multiple terabytes in size. To request access, contact Bruce Wittmann. In addition, summary statistics describing these raw data have been deposited at Caltech Data and are publicly available. DOIs are listed in Table A-11. This chapter analyzes existing, publicly available data. The accession numbers for the datasets are listed in Table A-11.

All original code has been deposited at Caltech Data and is publicly available. DOIs are listed in Table A-11.

Any additional information required to reanalyze the data reported in this chapter is available from Bruce Wittmann upon request.

A.2 Method Details

A.2.1 Alignment Generation and EVmutation Model Training

The EVcouplings webapp was used to both generate multiple sequence alignments (MSAs) as well as train the EVmutation model needed for zero-shot prediction.[195] The GB1 sequence (See Encoding Preparation, below) was passed into the EVcouplings webapp, and alignments were made against the UniRef100 database for bitscore inclusion thresholds of 0.30, 0.35, 0.40, 0.45, and 0.50. All other EVcouplings parameters were kept at their default values (Alignment threshold type = Bitscore; Search iterations = 5; Position filter = 70%; Sequence fragment filter = 50%; Removing similar sequences = 90%; Downweighting similar sequences = 80%). A bitscore threshold of 0.40 returned an alignment with the most redundancy-reduced sequences (56) that covered all variable positions in the GB1 landscape at ≥70% coverage (i.e., less than 30% of aligned sequences had gaps at the positions of interest). Bitscores of 0.30 and 0.35 returned more redundancy-reduced sequences (2427 and

664, respectively), but failed to cover position 54 in the landscape. Bitscores of 0.45 and 0.50 covered all positions, but returned less redundancy-reduced sequences (30 and 27, respectively). I decided to move forward with the alignment and EVmutation model generated at a bitscore of 0.40. The alignment (in .a2m format) and the parameters for the EVmutation model trained on it (the ".model" file) were downloaded from the EVcouplings webapp. The alignment downloaded would also be used to train a DeepSequence VAE as well as build encodings and make zero-shot predictions using the MSA Transformer (See A.2.2: Encoding Preparation, A.2.3.1: EVmutation/DeepSequence Calculations, and A.2.3.2: Mask Filling Protocol, all below).

A.2.2 Encoding Preparation

I investigated encoding using three different strategies: one-hot, physicochemical parameters, and learned embeddings. One-hot encodings were prepared by first assigning each amino acid an index. To encode each GB1 variant, a 4 × 20 ("N amino acids per combo" × "N possible amino acids") matrix filled with 0's was instantiated where the index of each row corresponded to the position index in the variant. Each row of the matrix was then populated with a single value of "1" at the index corresponding to the appropriate amino acid. Repeating this procedure for all GB1 variants with experimentally measured fitness yielded a 149,361 × 4 × 20 tensor representing the complete set of one-hot encodings.

Physicochemical encodings were prepared using the descriptors originally published by Georgiev, using the values found in code published by Ofer & Linial.[36,124] To encode all variants, a 160,000 × 4 × 19 tensor was instantiated ("N possible combos" × "N amino acids per combo" × "N Georgiev parameters"). Every possible variant was encoded using all 19 Georgiev parameters, and the encodings were stored in the instantiated tensor. The last two dimensions of the tensor were then flattened to produce a 160,000 × 76 matrix and each column of the matrix was mean-centered and unit-scaled. The final encoding tensor was generated by extracting only those rows belonging to GB1 variants with experimentally measured fitness, then reshaping the last dimension to produce a 149,361 × 4 × 19 tensor.

Learned embeddings were prepared using the pre-trained models published in TAPE, ESM, and ProtTrans GitHub repositories.[51,52,55,58,60,61] For the non-MSA Transformer models, embeddings were generated by first building fasta files containing the full amino acid sequences of all 160,000 possible GB1 variants at positions 39, 40, 41, and 54. The template sequence used was:

```
MQYKLILNGKTLKGETTTEAVDAATAEKVFKQYANDNGVDGEWTYDDATKTFTVTE
```

The sequences contained in the fasta file were then embedded using the appropriate code provided from each of the repositories. For the TAPE models ("Bepler," "ResNet," "UniRep," "TAPE-Transformer," and "LSTM") the `tape-embed` command from the software associated with the original publication by Rao et al. (https://github.com/songlab-cal/tape-neurips2019) was used to generate embeddings. For the ESM model (esm1b_t33_650M_UR50S), the example code provided in the "Quick Start" section of the GitHub repository (https://github.com/facebookresearch/esm#quick-start-) was used to generate embeddings. For the ProtTrans model ("ProtBert-BFD"), code from the example Jupyter notebook provided on the associated GitHub repository (https://github.com/agemagician/ProtTrans/blob/master/Embedding/PyTorch/Advanced/ProtBert-BFD.ipynb) was used to generate embeddings. Each of these processes generated tensors of shape 160,000 × S × L ("N possible combos" × "Tokenized Sequence Length" × "L latent dimensions"). The value of L varied by encoding used, and is given in Table A-1. The tokenized sequence length (S) varied depending on whether or not the embeddings for "<cls>" and "<eos>" tokens were returned alongside the embeddings for the different amino acids. The value of S was taken into account in the next step, where the embeddings corresponding to amino acids varied in the GB1 dataset (Using 0-indexing, this was indices 38, 39, 40, and 53 of the middle dimension of the output tensor if no <cls> token was added, and 39, 40, 41, and 54 if a <cls> token was added) were extracted from the output tensor. Using the same procedure as with the physicochemical properties, the resultant 160,000 × 4 × L tensor was mean-centered and unit-scaled to produce a 160,000 × 4L matrix. Finally, the appropriate rows were isolated and the last dimension reshaped to produce a final learned embedding encoding tensor of shape 149,361 × 4 × L.

Unlike all other models tested, the MSA Transformer takes an MSA as an input rather than a sequence. To build variant MSAs, I used the MSA generated by the EVcouplings webserver (See A.2.1: Alignment Generation and EVmutation Model Training) as a template. To begin, I filtered out all lowercase characters and the "." character for each sequence in the template MSA. Next, I removed any duplicate sequences from the filtered MSA. I used this filtered and de-duplicated MSA as a template to build an MSA for each mutant in the GB1 landscape. When building mutants, I changed only the GB1 reference sequence, keeping all other sequences in the MSA constant. The resultant 160,000-mutant MSAs were then streamed through the MSA Transformer, extracting the embeddings for the mutant GB1 positions in the first sequence of each alignment (corresponding to the mutant GB1 sequence). This procedure resulted in a 160,000 × 4 × 768 tensor (where "768" is the number of latent dimensions assigned to each token by the MSA Transformer). This tensor was then mean-centered and unit-scaled following the same procedure as for all other encodings before being filtered to produce a tensor of shape 149,361 × 4 × 768.

The encodings generated from the above procedures are made available at Caltech Data. Code provided on the MLDE repository enables replication of the encodings generated for the GB1 combinatorial landscape.

A.2.3 Zero-Shot Predictions

A.2.3.1 EVmutation/DeepSequence Calculations

EVmutation calculations were run using the model downloaded from the EVcouplings webserver (See A.2.1: Alignment Generation and EVmutation Model Training). This model had been trained on the MSA of the GB1 reference sequence generated against the UniRef100 database with a bitscore inclusion threshold of 0.40. The example code provided in the EVcouplings GitHub repository was used as a template for making zero-shot predictions of GB1 fitness using the downloaded EVmutation model (https://github.com/debbiemarkslab/EVcouplings/blob/develop/notebooks/model_parameters_mutation_effects.ipynb). Code is provided in the MLDE GitHub repository for replicating the predictions made in this chapter, as well as for applying EVmutation to any other combinatorial landscape.

To perform DeepSequence calculations, the associated variational autoencoder (VAE) first needed to be trained. Using the same MSA downloaded along with the EVmutation model, I trained the DeepSequence VAE using code from the DeepSequence GitHub repository (https://github.com/debbiemarkslab/DeepSequence/blob/master/examples/run_svi.py). The code was modified to allow passing in the GB1 MSA. Predictions were subsequently made using the trained VAE by following additional example code from the DeepSequence GitHub repository (https://github.com/debbiemarkslab/DeepSequence/blob/master/examples/Mutation%20Effect%20Prediction.ipynb). I used 2000 prediction iterations for making predictions. Code for training a DeepSequence VAE on any combinatorial landscape is provided in the MLDE GitHub. Code for making predictions using a trained DeepSequence VAE is also provided.

A.2.3.2 Mask Filling Protocol

Zero-shot predictions using a mask filling protocol were performed for all models provided in the ESM GitHub repository as well as the models ProtBert and ProtBert-BFD in the ProtTrans GitHub repository.[51,55,61] For each of these models, I tested both a naïve and conditional mask filling strategy for making zero-shot predictions of GB1 fitness. For GB1, both protocols model the probability

$$P(combo|s_{const}) = P(aa_{39}aa_{40}aa_{41}aa_{54}|s_{const}), \quad \text{Eq. A-1}$$

where s_{const} is the sequence of the non-varying positions in the combinatorial landscape and aa_x gives the identity of the amino acid at position x in combination "combo." The difference between the naïve and conditional probability protocols is how the probability $P(aa_{39}aa_{40}aa_{41}aa_{54}|s_{const})$ is calculated. Naïve probability assumes that variable positions are independent of one another, and so calculates

$$P(combo|s_{const}) = P(aa_{39}|s_{const})P(aa_{40}|s_{const})P(aa_{41}|s_{const})P(aa_{54}|s_{const}). \quad \text{Eq. A-2}$$

Conditional probability, in contrast, does not assume independence of the variable positions and instead directly solves the probability given by Eq. A-1 using the product rule of probability. Note that, for all non-MSA Transformer methods, the parent GB1 sequence was

used to define s_{const}, while for the MSA Transformer, the MSA used for EVmutation (after applying the same filtering and de-duplication procedure from Encoding Preparation, above) was used to define s_{const}. Only the variable positions in the reference sequence of this MSA were masked (rather than the full alignment column corresponding to variable positions), so the MSA Transformer maintained full access to the information provided by other sequences in the MSA.

The naïve mask filling protocol began by masking all variable positions in the GB1 reference sequence. This masked sequence (or masked alignment) was then passed into the model and the logits of the masked positions were extracted to yield a 4 × A matrix, where "A" is the alphabet size of the model being used. A softmax function was then applied over the alphabet dimension to yield the probability of each amino acid (and other special characters included in the alphabet, though the probability of these was vanishingly small) occurring at each position given the context of the non-varying GB1 sequence (e.g., the probability $P(aa_x|s_{const})$ for x=39,40,41 and 54, where the position along the first axis of the matrix corresponds to a given x); the log of the matrix was next taken to convert probability to log-probability. Finally, log-probability of a combination was calculated using

$$log\big(P(combo|s_{const})\big) = log\big(P(aa_{39}|s_{const})\big) + log\big(P(aa_{40}|s_{const})\big) + log\big(P(aa_{41}|s_{const})\big) + log\big(P(aa_{54}|s_{const})\big),$$

where s_{const} is the sequence of the non-varying positions in the combinatorial landscape and aa_x gives the identity of the amino acid at position x in combination "combo."

The conditional mask filling protocol allowed for dependencies between the amino acid identities at the variable positions. I calculated conditional probability by summing all possible factorizations of $P(combo|s_{const})$ using the product rule of probability. For instance, one possible factorization using the product rule is

$$\begin{aligned} P(combo|s_{const}) &= P(aa_{39}aa_{40}aa_{41}aa_{54}|s_{const}) \\ &= P(aa_{39}aa_{40}aa_{41}|aa_{54}s_{const})P(aa_{54}|s_{const}) \end{aligned}$$

$$= P(aa_{39}aa_{40}|aa_{41}aa_{54}s_{const})P(aa_{41}|aa_{54}s_{const})P(aa_{54}|s_{const})$$
$$= P(aa_{39}|aa_{40}aa_{41}aa_{54}s_{const})P(aa_{40}|aa_{41}aa_{54}s_{const})$$
$$P(aa_{41}|aa_{54}s_{const})P(aa_{54}|s_{const}).$$

In words, this factorization translates to (1) the probability of a specific amino acid at position 54 when all other variable positions are masked multiplied by (2) the probability of a specific amino acid at position 41 given the specific amino acid a position 54 but masking positions 39 and 40 multiplied by (3) the probability of a specific amino acid at position 40 given the specific amino acids at positions 41 and 54 but masking position 39 multiplied by (4) the probability of a specific amino acid at position 39 given the specific amino acids at positions 40, 41, and 54; all probabilities are calculated within the context of the remaining GB1 sequence. Of course, the order of factorization is arbitrary (for instance, in the first step I could have instead factored to $P(aa_{40}aa_{41}aa_{54}|aa_{39}s_{const})P(aa_{39}|s_{const})$), and any ordering of factorization can reconstruct the probability $P(aa_{39}aa_{40}aa_{41}aa_{54}|s_{const})$. There are 24 total factorizations possible (all permutations of the 4 variable positions), and the final conditional probability $P(aa_{39}aa_{40}aa_{41}aa_{54}|s_{const})$ was calculated by summing them all together.

Calculation of the conditional probability was much more expensive than the calculation of naïve probability. For instance, while determination of naïve probability for GB1 required calculating just the variable-position logits of the GB1 sequence with variable positions masked (a single pass through a model), calculation of conditional probability required calculating the variable-position logits of all 34,481 possible masked combinations. While trivial for smaller models, calculating conditional probability using larger models with complex transformations (such as the MSA Transformer or ESM1b) could become expensive. Regardless, calculation of component probabilities used for calculating the overall conditional probably was performed in the same way as for naïve probability. For each component probability: (1) appropriate positions in the GB1 sequence were masked, (2) a softmax was taken over the alphabet dimension, and (3) the appropriate probabilities were extracted from the resultant probability matrix. Note that, unlike for naïve calculation, I did not take the log of the calculated probability matrices—the necessity of calculating a sum

over the different factorizations precluded this possibility. While such a practice may lead to numerical instability for large combinatorial libraries, I observed no such problems for GB1.

Code for calculating the naïve and conditional probabilities of any combinatorial library using any model evaluated in this chapter is provided in the MLDE package.

A.2.3.3 ΔΔG Calculations

ΔΔG calculations were performed using a local copy of the Triad software suite (version 2.1.1, Protabit, Pasadena, CA, USA: https://triad.protabit.com/). To begin, the template protein crystal structure (PDB: 2GI9) was prepared for calculations via the below command:

```
$ ~/triad-2.1.2/triad.sh ~/triad 2.1.2/apps/preparation/proteinProcess.py struct 2GI9.pdb --crosetta
```

This command generated two files: 2GI9_process.pdb and 2GI9_prepared.pdb.[154] The "_process" pdb file is the 2GI9.pdb file prepared for downstream Triad calculations but without any structural minimization. The "_prepared" pdb file is the 2GI9.pdb file prepared for downstream Triad calculations but with an added constrained minimization. The flexible backbone calculations were run using the standard Rosetta scoring function and the "_prepared" pdb file. The command line call is below:

```
$ ~/triad-2.1.2/tools/openmpi/bin/mpirun  np 96 ~/triad 2.1.2/triad.sh ~/triad 2.1.2/apps/cleanSequences_BjwMod.py  struct ./2GI9_prepared.pdb  inputSequences 2GI9.mut  crosetta  calculateRmsd  minDesign  inputSequenceFormat  pid floatNearbyResidues 2>&1 | tee $OUTPUT
```

Note that the cleanSequences_BjwMod.py file is a version of the inbuilt Triad script cleanSequences.py modified to also output root mean squared deviation (RMSD) of the protein backbone. The 'inputSequences' file '2GI9.mut' describes the mutations for all 149,360 GB1 variants relative to the parent GB1 protein. The fixed backbone calculations were run with the "_process" pdb file using a Rosetta scoring function that has a Van der Waals term with a softer inner wall, reducing the chance that steric clashes produce overly high energies. The command line call for the flexible backbone calculations is below:

```
$ ~/triad-2.1.2/tools/openmpi/bin/mpirun  np 12 ~/triad 2.1.2/triad.sh ~/triad 2.1.2/apps/cleanSequences.py  struct ../pdbs/2GI9_process.pdb  inputSequences ../2GI9.mut  rosetta  inputSequenceFormat pid  floatNearbyResidues  soft 2>&1 | tee $OUTPUT
```

There was no output file directly produced by the cleanSequence.py script, hence the captured output. The captured output file generated was parsed to extract ΔG values for each protein variant, which were in turn used to calculate ΔΔG values relative to the parent protein. In this work, I defined a negative ΔΔG to be stabilizing and a positive ΔΔG to be destabilizing relative to the parent protein; this necessitated flipping the sign of the literature ΔΔG values when building Figure A-2, as Nisthal et al. defined opposite meanings of the sign of ΔΔG.[153]

A.2.4 Simulation Details

A.2.4.1 Encoding Comparison Simulations

The simulation procedure for comparing encoding strategies was designed to enable pairwise comparison of simulation results using the different encodings. For a given simulation, each of the tested encodings shared the same training set (same variant identities), cross-validation indices, and random seeds used to initialize models that rely on randomness for training.

All encoding comparison simulations were run with a randomly drawn training set of 384, 48, or 24 variants (drawn without replacement) and five-fold cross-validation. Some model classes scale poorly with large input spaces, and so, due to computational expense, not all inbuilt MLDE models were used when encoding using learned embeddings from the TAPE transformer, the MSA Transformer, ESM1b, ProtBert-BFD, UniRep, and the TAPE LSTM. For these encoding strategies, when training size was 384, the sklearn RandomForestRegressor, sklearn BaggingRegressor, and sklearn KNeighborsRegressor classes were omitted from the ensemble of models trained. When training size was 24 or 48, the sklearn ARDRegression, sklearn BaggingRegressor, and sklearn KNeighborsRegressor classes were omitted from the ensemble of models trained. All other simulations for the other encodings were performed using all 22 inbuilt MLDE models (See A.3.1: Inbuilt Models for architectures). Trained models were then ranked according to their cross-validation mean squared error (MSE) and the predictions of the top three were averaged to predict the fitness

of the remaining variants (See A.3: MLDE Programmatic Implementation for details on model averaging). The values of NDCG, max achieved fitness, and mean achieved fitness reported for this set of simulations are all based on these predictions.

A.2.4.2 High-Fitness Simulations

A given training set of designed high-fitness training data was produced by sampling variants such that 50% had a fitness greater than or equal to the value of given threshold and 50% had fitness below. Additionally, it was enforced that no variant with fitness greater than 0.34 could be chosen. Threshold values of 0.011, 0.034, 0.057, and 0.080 were used in this study to design fitness-enriched training sets.

To generate multiple training sets for a given threshold, the GB1 dataset was first filtered to exclude all variants with fitness greater than 0.34. The remaining data was then split into two sets: one set had all variants with fitness greater than or equal to the threshold and the other set had all variants with fitness less than the threshold. Equal numbers of samples were then drawn at random from the two sets without replacement. Training data for the "no-threshold" control discussed in Section 2.2 and presented in Figure 2-3C-E ("100% Training Fitness ≥ 0") was generated by sampling at random from the GB1 dataset filtered to exclude variants with fitness greater than 0.34.

For each of the four thresholds and the no-threshold control, 2000 training sets were generated, each containing 384 variants (10,000 training sets of 384 variants in total). For simulations using 24 or 48 training samples, the first 24 or 48 variants in these training sets, respectively, were used for training. Each training set was then fed into the simulated MLDE pipeline using Georgiev parameters for variant encoding and 5-fold cross-validation for model selection. For the sake of computational efficiency, only CPU-bound models (those from scikit-learn and XGBoost) were trained for these simulations. Trained models were then ranked according to their cross-validation MSE and the predictions of the top three were averaged to predict the fitness of the unlabeled variants. The values of NDCG, max achieved fitness, and mean achieved fitness reported for this set of simulations are all based on these predictions.

A.2.4.3 Zero-Shot Simulations

To generate training data using a zero-shot predictor, the GB1 dataset was first ranked by the zero-shot predictions. Next, the top 1600 variants (i.e., the 1600 variants predicted to have the highest fitness) were identified, and 2000 random samples of 384 were drawn at random without replacement. This process was repeated for the top-ranked 3200, 6400, 9600, 12,800, 16,000, and 32,000 variants, resulting in 14,000 total training sets per zero-shot predictor, each containing 384 random samples, for 42,000 training sets in total (3 zero-shot strategies × 14,000 training sets/zero-shot strategy = 42,000 total training sets).

For the 384-training-sample simulations, each of the 42,000 training sets was then fed into the simulated MLDE pipeline. Again, for the sake of computational efficiency, only CPU-bound (scikit-learn and XGBoost) models were evaluated. Simulations were performed using one-hot, Georgiev parameters, and learned embeddings from the MSA Transformer for encoding with 5-fold cross-validation for determining model effectiveness. As with the encoding comparison simulations, the models from the sklearn classes RandomForestRegressor, BaggingRegressor, and KNeighborsRegressor were omitted when encoding using the MSA Transformer due to poor scaling with input feature size. Trained models were ranked according to their cross-validation MSE and the predictions of the top three were averaged to predict the fitness of the remaining variants. Only the top-predicted unsampled combinations that could be constructed by recombining combinations in the training data were evaluated, enforcing a confidence threshold on predictions and focusing all resources on regions believed to contain the highest-fitness protein variants. The reported values of max achieved fitness and mean achieved fitness reported for this set of simulations are all derived from this restricted set of evaluated proteins, though the fitness value returned is still normalized to the full unsampled set. For a given simulation, the global maximum is considered to be achieved if it is present in either the training data or the evaluated predictions. The random controls presented in the Section 2.2 and in Figure 2-5 and Figure 2-6 are derived from the Encoding Comparison Simulations, but only evaluating the CPU models and employing the same confidence threshold strategy for evaluating predictions.

For the 24- and 48-training-sample simulations, the first 24 and 48 variants in each of the full 384-sample training sets were used for training, respectively. The models omitted from evaluation for MSA Transformer here were given by the sklearn classes ARDRegression, BaggingRegressor, and KNeighborsRegressor. Otherwise, the procedure was the same as for the 384-training-sample simulations.

A.2.4.4 Traditional Directed Evolution Simulations

Traditional DE simulations were performed from every variant in the GB1 landscape with non-zero starting fitness. Zero-fitness variants were omitted from these simulations as a researcher would never begin a DE study from such a variant. As in the MLDE simulations, variants with imputed fitness in the GB1 dataset were ignored for these simulations.

A greedy walk simulation begins with 4 potential positions to evaluate. One of these positions is selected, the fitness values of all amino acids at this position are evaluated, and the best mutation is fixed. In the next round, there are three positions to evaluate. One of these positions is selected, all mutants are evaluated, and the best mutation is fixed again. This process continues until all positions have been evaluated; the fitness of the best variant identified in the last round is returned. The results reported for the greedy walk simulations consider all possible paths from all non-zero-fitness starting variants (with 24 paths per starting variant and 119,814 non-zero fitness starting points, this is 2,877,216 simulated greedy walks in total).

A.2.5 Evaluation Metrics

The evaluation metrics used in the work from Chapter 2 include (1) the max normalized true fitness of the M-highest-ranked variants, (2) the mean normalized true fitness of the M-highest-ranked variants, and (3) the ranking metric “normalized discounted cumulative gain” (NDCG) of all predictions (where “gain” is defined as the normalized fitness of unsampled variants). NDCG was calculated using scikit-learn’s `ndcg_score()` function, which uses the form

$$NDCG = \left(\sum_{i=1}^{N} \frac{f_i}{log_2(i+1)}\right) \Bigg/ \left(\sum_{i=1}^{N} \frac{f_i'}{log_2(i+1)}\right),$$

where f is the true fitness of all (N) unsampled variants ranked by predicted fitness and f' is the true fitness of all unsampled variants ranked by true fitness (i.e., the ideal ordering). When evaluating a single MLDE simulation, the fitness was normalized to the highest-fitness variant in the unsampled data. Typically, this was equivalent to normalizing to the highest fitness in the entire GB1 dataset, as it was extremely unlikely that the highest-fitness variant in the dataset was drawn in the training set. Still, normalizing to the highest unsampled fitness allowed me to make more fair comparisons between MLDE simulations in the rare case that the highest-fitness value appeared in the training data.

A.3 MLDE Programmatic Implementation

The MLDE algorithm takes as input all encodings corresponding to the combinations of amino acids found in the training data along with their measured fitness values. During the training stage, these sampled combinations are used to train a version of all inbuilt model architectures (A.3.1: Inbuilt Models). Specifically, k-fold cross-validation is performed to train each model using the default model parameters; mean validation error from the k-fold cross-validation (mean squared error) is recorded for each architecture. All model instances trained during k-fold cross-validation are also stored for later use. For instance, if evaluating all 22 inbuilt model architectures with 5-fold cross-validation, 22 × 5 = 110 total trained model instances are recorded. For making predictions, the top N model architectures (those with the lowest cross-validation error) are first identified. For each of the top N model architectures, predictions are made on the unsampled combinations by averaging the predictions of the k × N model instances stored during cross-validation. For instance, if testing the top 3 model architectures identified from 5-fold cross-validation, this means that the predictions of 3 × 5 = 15 total models (3 architectures × 5 model instances/architecture saved during cross-validation) are used for prediction. For all simulations presented in Chapter 2, I evaluated model architectures using 5-fold cross-validation and then made

predictions using the top 3 (again, meaning that the predictions of 3 × 5 = 15 total models were averaged for each simulation).

A.3.1 Inbuilt Models

A.3.1.1 Keras

Five separate neural network architectures were implemented using the Python package Keras: three fully connected neural network architectures and two 1D-convolutional neural network architectures. The identities and default values of tunable hyperparameters are given in Table A-8. All neural networks were trained with a batch size of 32 using the "adam" optimizer for at most 1000 epochs with early stopping after 10 epochs with no improvement in validation error (calculated against the cross-validation test data).

The fully connected neural networks differed in the number of hidden layers: zero, one, or two. After each hidden layer, a batch normalization layer was employed, followed by an exponential linear unit (ELU) nonlinearity. A single dropout layer was used before the output layer. The output layer was a scalar value passed through an ELU nonlinearity.

The convolutional neural networks differed in the number 1D convolutional layers: one or two. After each convolutional layer, a batch normalization layer was employed followed by an ELU nonlinearity. Following the convolutional layers, the output matrix was flattened with a GlobalAveragePooling1D layer. After flattening, a single dropout layer was used before the output layer. The output layer was a scalar value passed through an ELU nonlinearity.

A.3.1.2 XGBoost

Four gradient boosting approaches were implemented in MLDE using the Python package XGBoost:[126] both tree and linear base models were implemented with both "reg:squarederror" and "reg:tweedie" objectives. For reg:tweedie, `tweedie_variance_power` was set to 1.5. The identities and default values for tunable XGBoost hyperparameters are given in Table A-9. Unless explicitly mentioned in Table A-9, all XGBoost parameters were held at their default values as detailed in the official XGBoost documentation. The descriptions for all parameters can also be found in the official XGBoost

documentation. All XGBoost models used in this work were implemented with early stopping: `eval_metric` was set to “rmse” when the “reg:squarederror” objective was used and “tweedie-nloglik@1.5” when the “reg:tweedie” objective was used; validation error was calculated against the cross-validation test data; training was terminated if validation error did not decrease for 10 epochs or 1000 total training epochs had passed.

A.3.1.3 Scikit-learn

Only scikit-learn models were used in the original implementation of MLDE.[82,196] To remain consistent with this first implementation, the regressor models from scikit-learn that were procedurally effective (i.e., those that did not consistently error during operation or else take a very long time to train) while using default parameters in this previous implementation were also used in the version presented in Chapter 2. The identities and default values for tunable scikit-learn hyperparameters are given in Table A-10. Unless explicitly mentioned in Table A-10, all scikit-learn parameters were held at their default values as detailed in the official scikit-learn documentation. The descriptions for all parameters can also be found in the official documentation.

A.4 Compute Environment

All MLDE code is written in Python using Anaconda as the environment manager. The Anaconda environments "mlde.yml" and “mlde2.yml” within the MLDE GitHub page can be used to build environments in which MLDE is known to be stable.

A.5 Computational Hardware Information

Simulations were performed across three workstations, a local server, and an r5.24xlarge Amazon Web Services (AWS) EC2 instance. All three workstations ran on Ubuntu 18.04.3 LTS. Two of the workstations contained Intel i7-8700 processors with an NVIDIA Titan V and NVIDIA GeForce RTX 2070 GPU; the third workstation contained an AMD Ryzen 9 3900x processor with two NVIDIA RTX 2070 GPUs. The local server ran on Ubuntu 20.04.02 LTS and contained 2 × Intel Xeon Gold 6248R processors; there were no GPUs in the local server. Triad calculations were performed on an Intel-containing desktop (fixed backbone) and a c5.24xlarge AWS EC2 instance (flexible backbone). Information regarding

which computer ran which specific simulation/computation with which specific piece of hardware is available upon request.

A.6 Supplementary Item Descriptions

Data S1. Encoding comparison summary metrics, related to Figure 2-2. This file provides summary statistics for the encoding comparison simulations from Section 2.2.2. For each encoding and training set size tested, I provide the mean, median, and standard deviation of five different summary metrics. The summary metrics are the "max fitness achieved," "mean fitness achieved," "NDCG," "NDCG-1536," and "NDCG-384." The summary metrics "NDCG-1536" and "NDCG-384" are the NDCG score calculated using only the 1536 and 384 highest-ranked samples, respectively.

Data S2. Pairwise encoding comparison results, related to Figure 2-2. Because the cross-validation indices, random seeds, and training datasets were kept the same for each encoding strategy tested in Section 2.2.2, pairwise comparisons of simulation results can be made. This file provides a pairwise comparison of simulation results by summary metric and training set size for each encoding tested in the Section 2.2.2. Specifically, it provides the counts and frequencies by which a given encoding using a given training set size achieved a higher summary metric score for the five different summary metrics provided in Data S1.

Data S3. Summary statistics for ftMLDE simulations compared to traditional DE simulations and MLDE simulations, related to Figure 2-5 and Figure 2-6. Summary statistics are provided for every simulation depicted in Figure 2-5 and Figure 2-6 as well as Figure A-7 – Figure A-10. Specific summary metrics are the "max fitness achieved," "mean fitness achieved," "NDCG," "NDCG-1536," "NDCG-384," "GlobalMaxFound," and "UnsampledMaxFound." The summary metrics "NDCG-1536" and "NDCG-384" are the NDCG score calculated using only the 1536 and 384 highest-ranked samples, respectively. The summary metric "GlobalMaxFound" is the percent of simulations in which the global maximum was identified in either the training data or tested predictions. The summary metric "UnsampledMaxFound" is the percent of simulations in which the maximum of the unsampled data was identified in the tested predictions. The final column

("BetterThanTrad") is "TRUE" if the combination of zero-shot strategy, encoding, sampling threshold, and training set size achieved the global maximum more frequently than traditional DE simulations.

A.7 Supplemental Figures

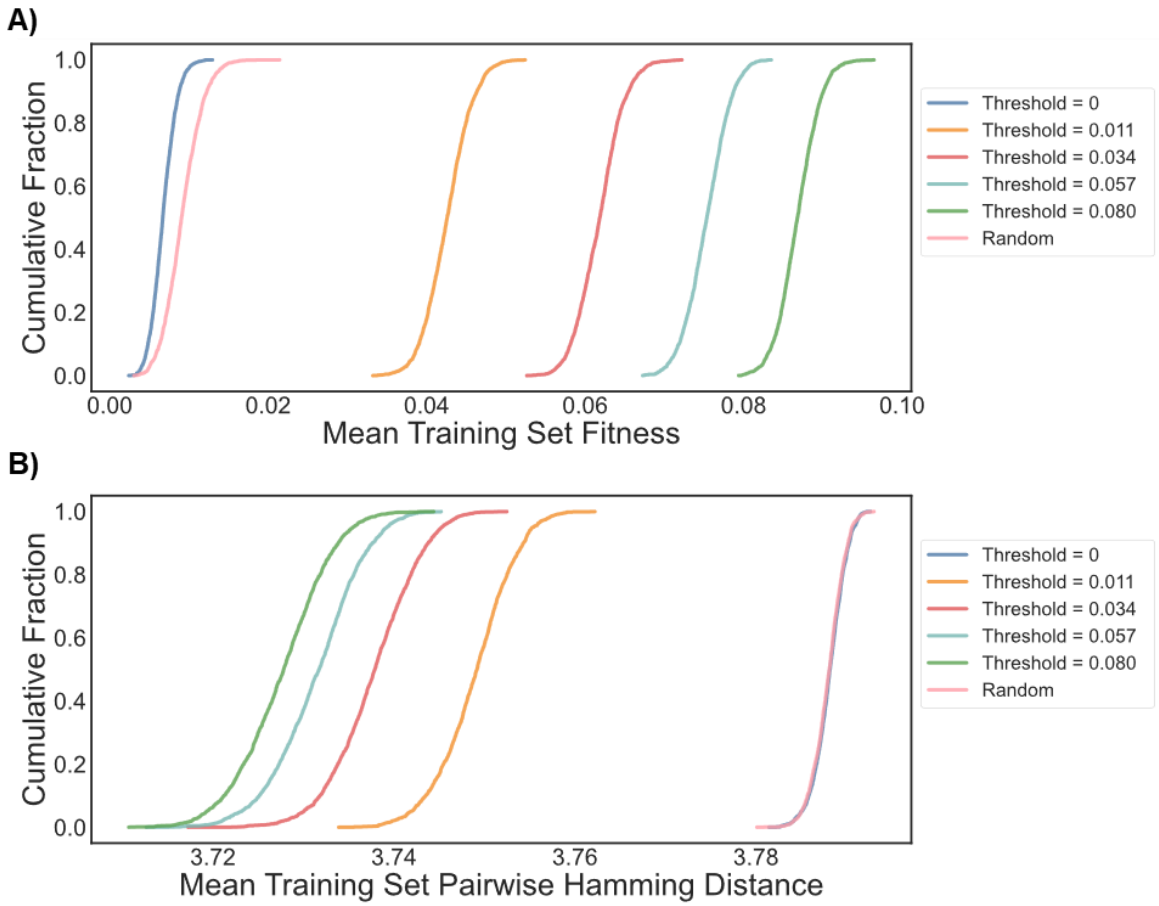


Figure A-1. Summary statistics (shown as empirical cumulative distribution functions) for the 2000 training sets (each consisting of 384 samples) designed to be enriched in fit variants, related to Figure 2-3. For a given threshold, 50% of variants in the training data have fitness greater than or equal to the threshold and the remainder have fitness less than or equal to the threshold. When the threshold = 0, 100% of variants in the training data have fitness greater than or equal to 0. For all thresholds (including the one with a threshold at 0), the maximum allowed fitness in the training data is 0.34. The random sample is equivalent to the data with a threshold at 0, but does not have the upper bound on training fitness. (A) Plots of the mean fitness of all variants in a training set for all 2000 training sets designed to be enriched in fitness. As the fitness threshold increases, the mean fitness rises as expected. (B) Plots of the mean pairwise hamming distance between all members of a training set for all 2000 training sets designed to be enriched in fitness. A higher mean pairwise hamming distance indicates greater sequence diversity in the training data. As the fitness threshold increases, the mean pairwise hamming distance decreases. This is because the training data is increasingly restricted to the narrow regions of sequence space that contain higher-fitness variants.

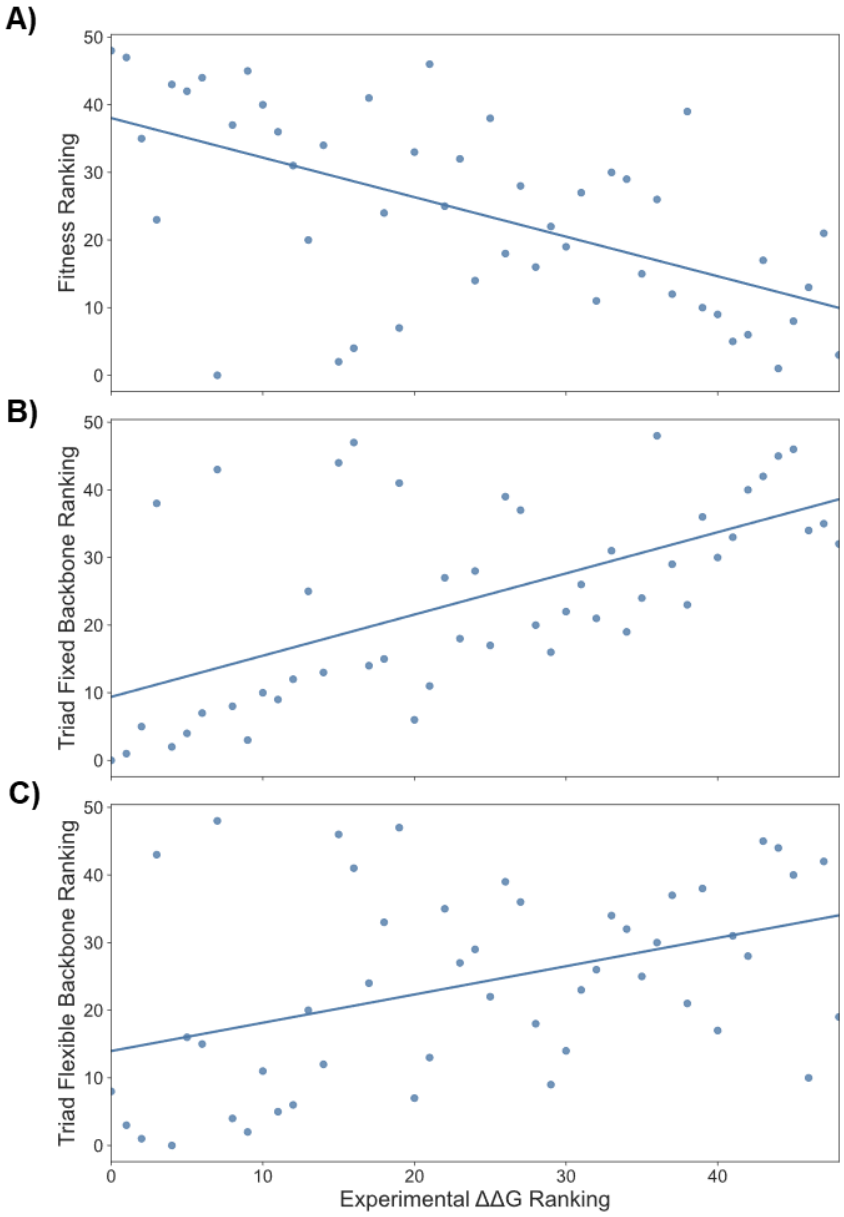


Figure A-2. Experimental ΔΔG versus true fitness and predicted ΔΔG, related to Figure 2-4. (A) Relationship between experimentally determined ΔΔG and GB1 fitness for single mutants at positions V39, D40, G41, and V54; both metrics are ranked from lowest value to highest value. I define a lower ΔΔG to be stabilizing and a higher ΔΔG to be destabilizing. The fitness of GB1 (at least at the considered positions) is loosely correlated with ΔΔG, but is clearly not the only determinant, with some lower-fitness variants having low ΔΔG and some higher-fitness variants having high ΔΔG. (B) Comparison of predicted ΔΔG upon mutation for GB1 variants using fixed backbone calculations to experimentally measured values of ΔΔG for single mutants at positions V39, D40, G41, and V54. (C) Comparison of predicted ΔΔG for GB1 variants using flexible backbone calculations to experimentally measured values of ΔΔG for single mutants at positions V39, D40, G41, and V54.

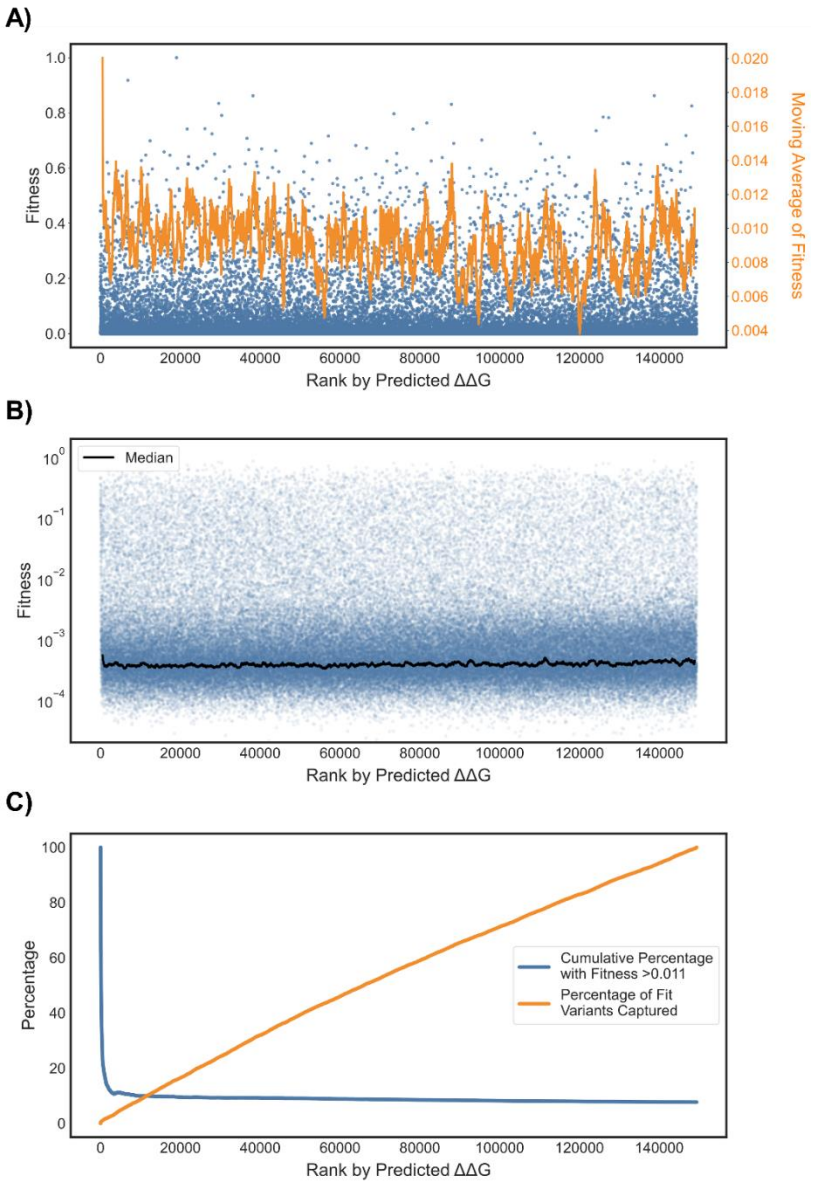


Figure A-3. Results of zero-shot prediction using flexible backbone Triad ΔΔG calculations, related to Figure 2-4. (A) The fitness of all GB1 variants plotted against the rank (from lowest to highest ΔΔG) given by Triad calculations. Blue dots are all individual variants while the orange line is the sliding mean (window size = 1000) of fitness. (B) The log-fitness of all GB1 variants plotted against the rank given by Triad calculations. Blue dots are all individual variants while the black line is the sliding median (window size = 1000) of fitness. (C) Cumulative fitness metrics for all GB1 variants ranked by Triad score. The blue curve gives the percentage of variants ranked up to and including a given Triad rank that have fitness greater than 0.011. The orange curve gives the percentage of all "fit" (defined as fitness greater than 0.011) variants encompassed in the set up to and including a given Triad rank.

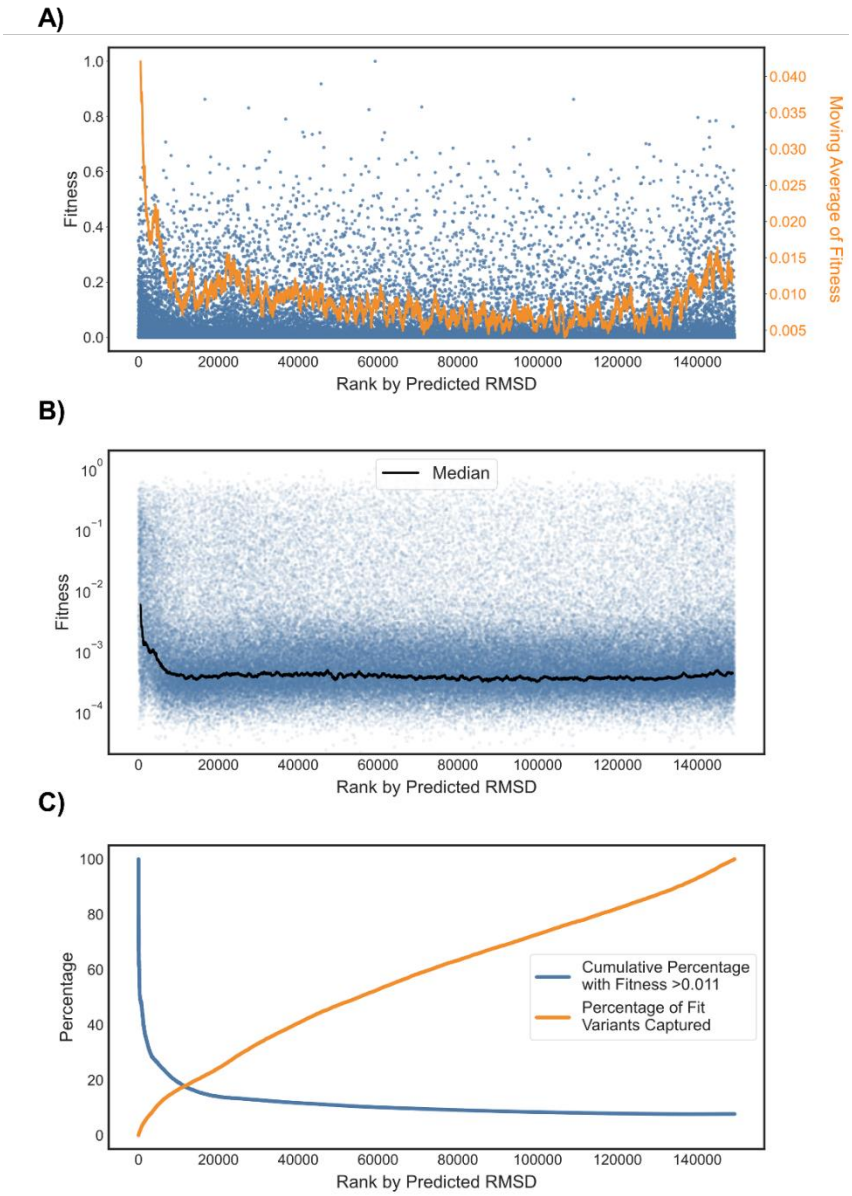


Figure A-4. Results of zero-shot prediction using flexible backbone Triad root mean squared deviation (RMSD) calculations, related to Figure 2-4. (A) The fitness of all GB1 variants plotted against the rank (from lowest to highest RMSD) given by Triad calculations. Blue dots are all individual variants while the orange line is the sliding mean (window size = 1000) of fitness. (B) The log-fitness of all GB1 variants plotted against the rank given by Triad calculations. Blue dots are all individual variants while the black line is the sliding median (window size = 1000) of fitness. (C) Cumulative fitness metrics for all GB1 variants ranked by Triad score. The blue curve gives the percentage of variants ranked up to and including a given Triad rank that have fitness greater than 0.011. The orange curve gives the percentage of all "fit" (defined as fitness greater than 0.011) variants encompassed in the set up to and including a given Triad rank.

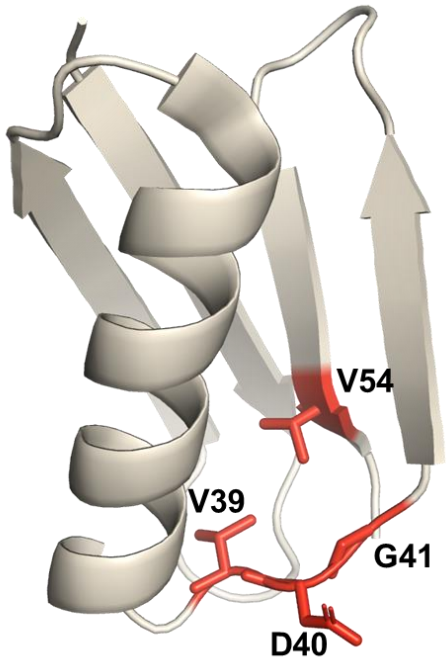


Figure A-5. GB1 crystal structure (PDB: 2GI9) with the positions mutated in the GB1 combinatorial landscape highlighted in red, related to Figure 2-4.

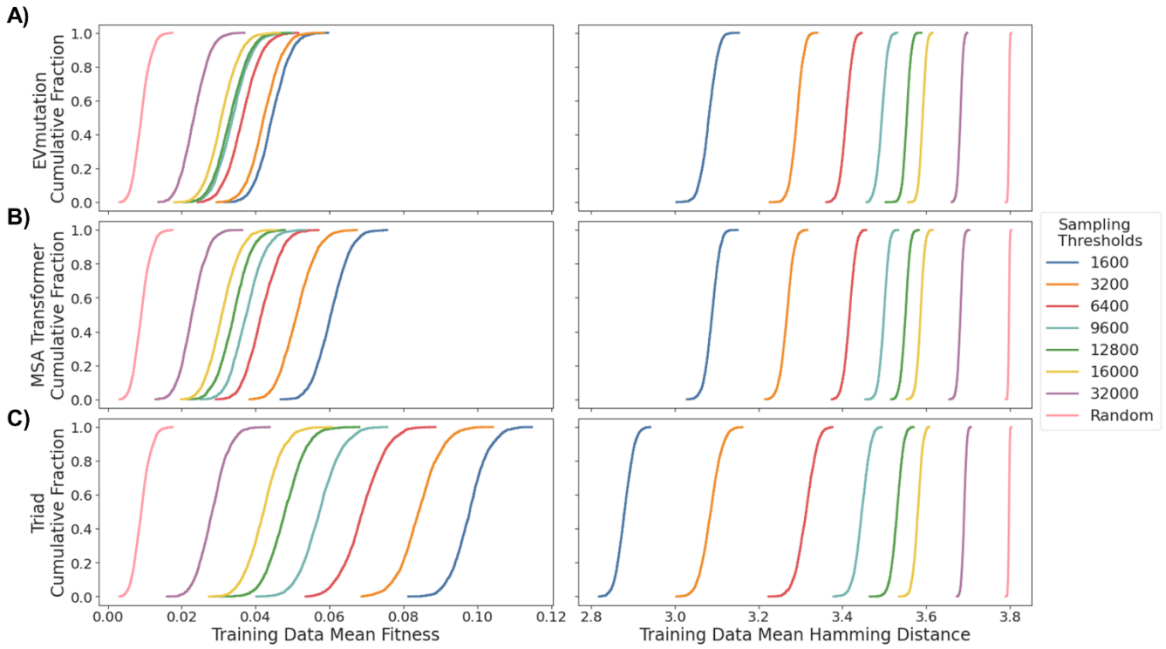


Figure A-6. Summary statistics (shown as empirical cumulative distribution functions) for the 384-sample training sets generated using all zero-shot predictors, related to Figure 2-5 and Figure 2-6. For each subplot (A–C), the left panel shows the mean fitness of all variants in a training set for all 2000 training sets derived for each sampling threshold; the right panel shows the mean pairwise hamming distance between all members of a training set for all 2000 training sets derived from each sampling threshold. In all subplots, as the threshold increases, the mean fitness decreases as more low-fitness variants have the potential to be included in the training data. Likewise, as the threshold increases, the mean pairwise hamming distance also increases. This is because predictive algorithms will tend to group similar sequences as having similar properties, and so sequences close in rank-order (as given by the zero-shot predictors) will be similar. By increasing the range of the rank from which we sample, the range of sequences sampled is thus also increased. (A) The summary statistics for training data derived from EVmutation zero-shot predictions. (B) The summary statistics for training data derived from zero-shot predictions made using the MSA Transformer for masked token prediction. (C) The summary statistics for training data derived from zero-shot predictions made using predicted ΔΔG with a fixed backbone.

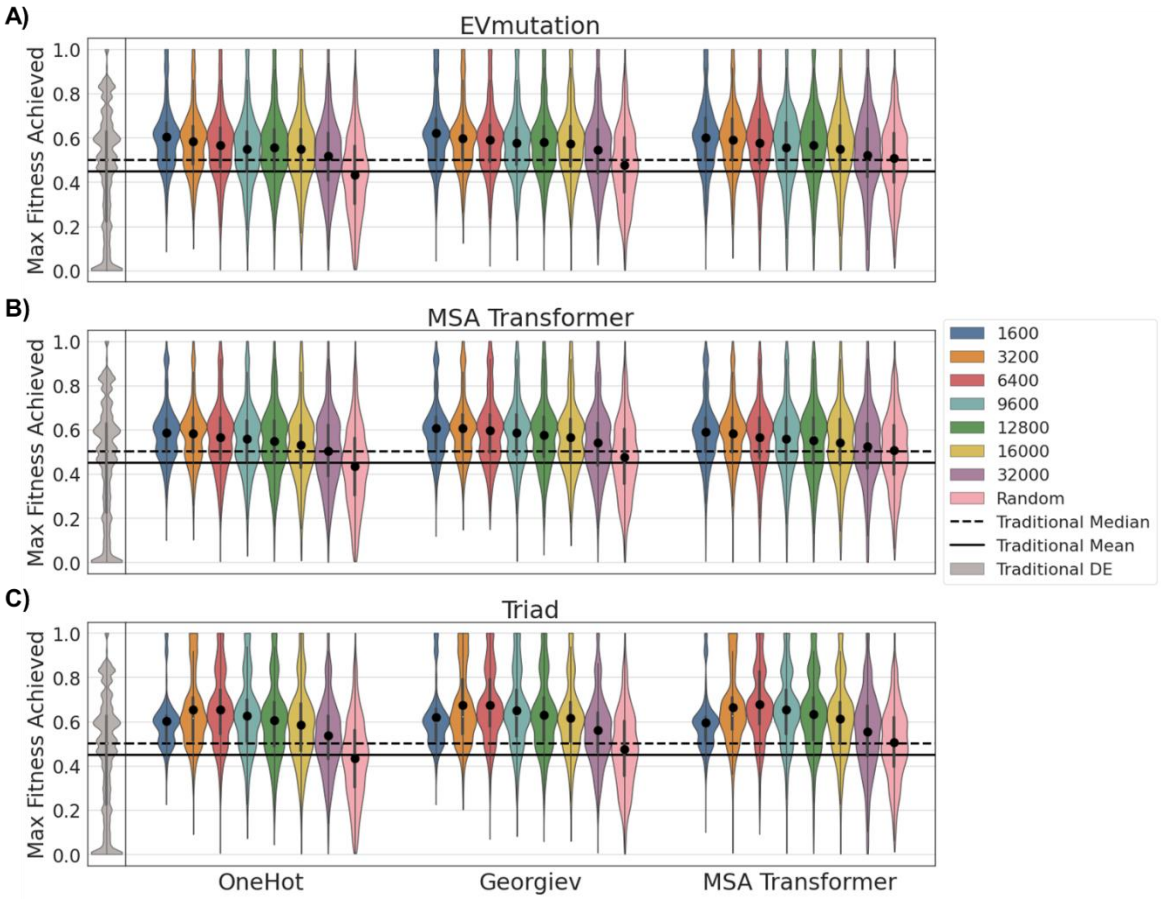


Figure A-7. Zero-shot prediction for training set design enables highly effective ftMLDE on the GB1 landscape, as measured by maximum fitness achieved in simulated experiments, related to Figure 2-5. Each subplot (A–C) shows the effect of different zero-shot predictors on the maximum fitness achieved in simulated ftMLDE experiments. Each violin (except for the grey ones corresponding to simulated traditional DE) represents data from 2000 simulated experiments where 48 variants were used for training and the top 32 predictions were tested. The major groupings of violins within each subplot correspond to different encoding strategies (one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer). The color of each violin corresponds to the zero-shot sampling threshold (i.e., the number of best-ranked variants according to a zero-shot predictor from which random samples were drawn to generate training data). Results of ftMLDE are compared to the results of simulated traditional DE (at the left of each plot, in grey) and standard MLDE (the three pink violins in each plot). (A) The maximum fitness achieved by simulated ftMLDE when EVmutation was used as the zero-shot predictor for training set design. (B) The maximum fitness achieved by simulated ftMLDE when a mask-filling protocol using the MSA Transformer was used as the zero-shot predictor for training set design. (C) The maximum fitness achieved by simulated ftMLDE when predicted $\Delta\Delta G$ was used as the zero-shot predictor for training set design.

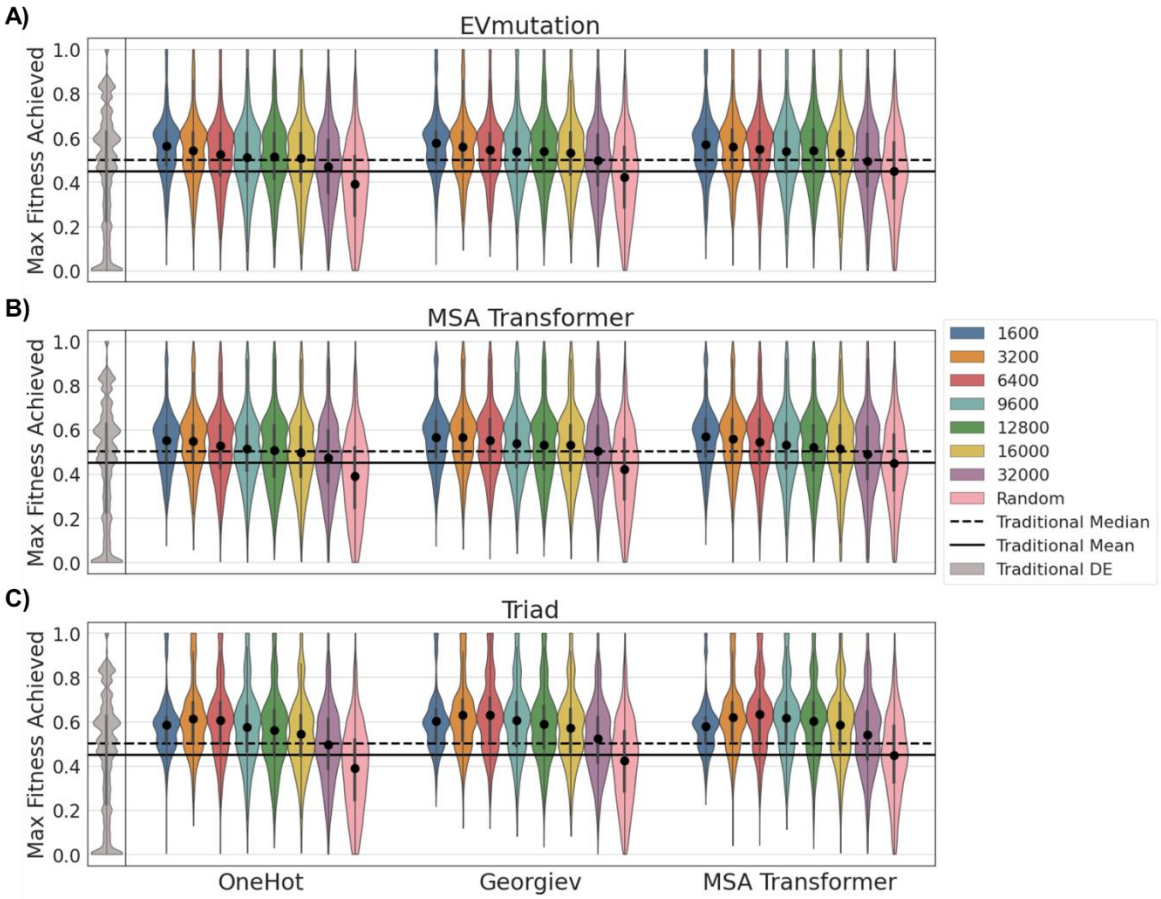


Figure A-8. Zero-shot prediction for training set design enables highly effective ftMLDE on the GB1 landscape, as measured by maximum fitness achieved in simulated experiments, related to Figure 2-5. Each subplot (A–C) shows the effect of different zero-shot predictors on the maximum fitness achieved in simulated ftMLDE experiments. Each violin (except for the grey ones corresponding to simulated traditional DE) represents data from 2000 simulated experiments where 24 variants were used for training and the top 56 predictions were tested. The major groupings of violins within each subplot correspond to different encoding strategies (one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer). The color of each violin corresponds to the zero-shot sampling threshold (i.e., the number of best-ranked variants according to a zero-shot predictor from which random samples were drawn to generate training data). Results of ftMLDE are compared to the results of simulated traditional DE (at the left of each plot, in grey) and standard MLDE (the three pink violins in each plot). (A) The maximum fitness achieved by simulated ftMLDE when EVmutation was used as the zero-shot predictor for training set design. (B) The maximum fitness achieved by simulated ftMLDE when a mask-filling protocol using the MSA Transformer was used as the zero-shot predictor for training set design. (C) The maximum fitness achieved by simulated ftMLDE when predicted ΔΔG was used as the zero-shot predictor for training set design.

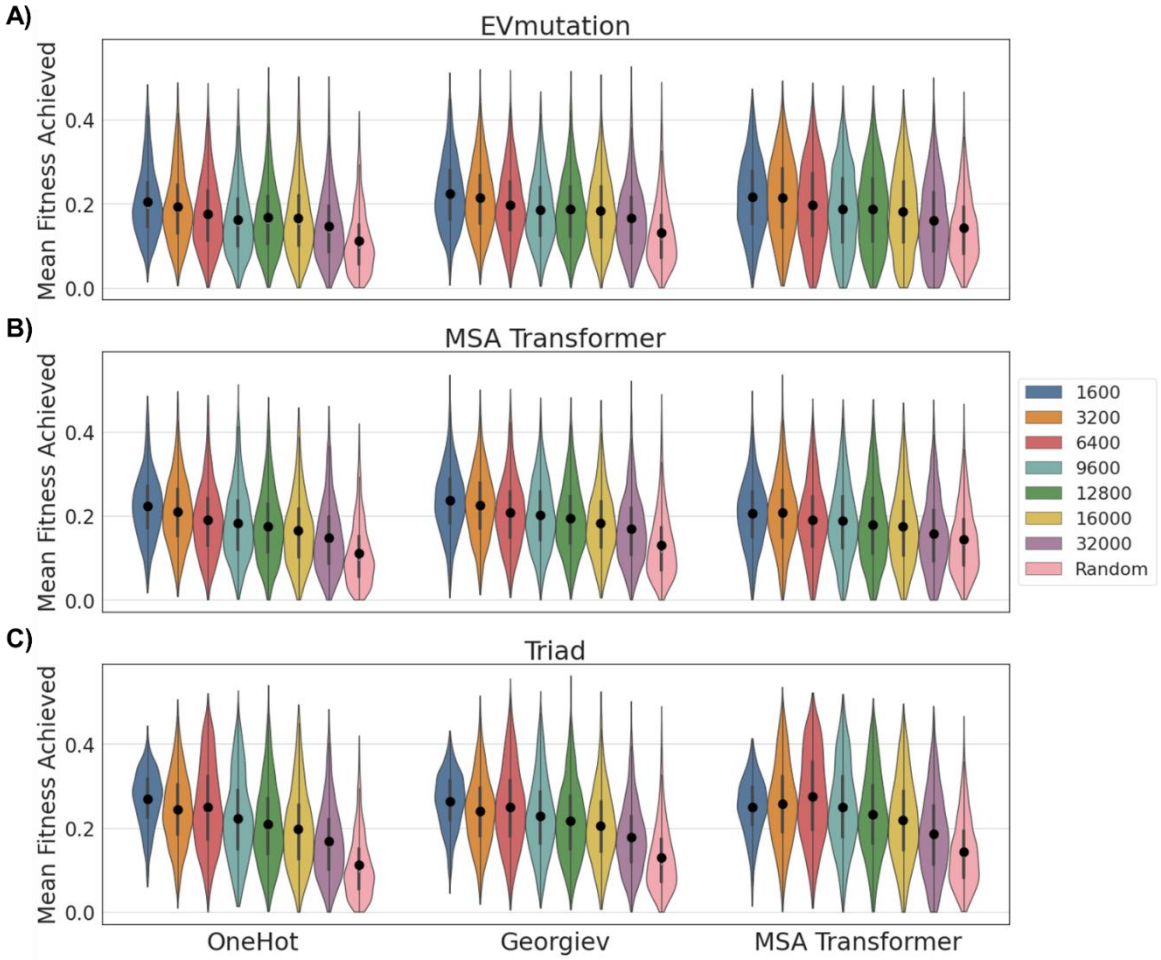


Figure A-9. Zero-shot prediction for training set design enables highly effective ftMLDE on the GB1 landscape, as measured by mean fitness achieved in simulated experiments, related to Figure 2-6. Each subplot (A–C) shows the effect of different zero-shot predictors on the mean fitness achieved in simulated ftMLDE experiments. Each violin represents data from 2000 simulated experiments where 48 variants were used for training and the top 32 predictions were tested. The major groupings of violins within each subplot correspond to different encoding strategies (one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer). The color of each violin corresponds to the zero-shot sampling threshold (i.e., the number of best-ranked variants according to a zero-shot predictor from which random samples were drawn to generate training data). Results of ftMLDE are compared to the results of standard MLDE (the three pink violins in each plot). (A) The mean fitness achieved by simulated ftMLDE when EVmutation was used as the zero-shot predictor for training set design. (B) The mean fitness achieved by simulated ftMLDE when a mask-filling protocol using the MSA Transformer was used as the zero-shot predictor for training set design. (C) The mean fitness achieved by simulated ftMLDE when predicted $\Delta\Delta G$ was used as the zero-shot predictor for training set design.

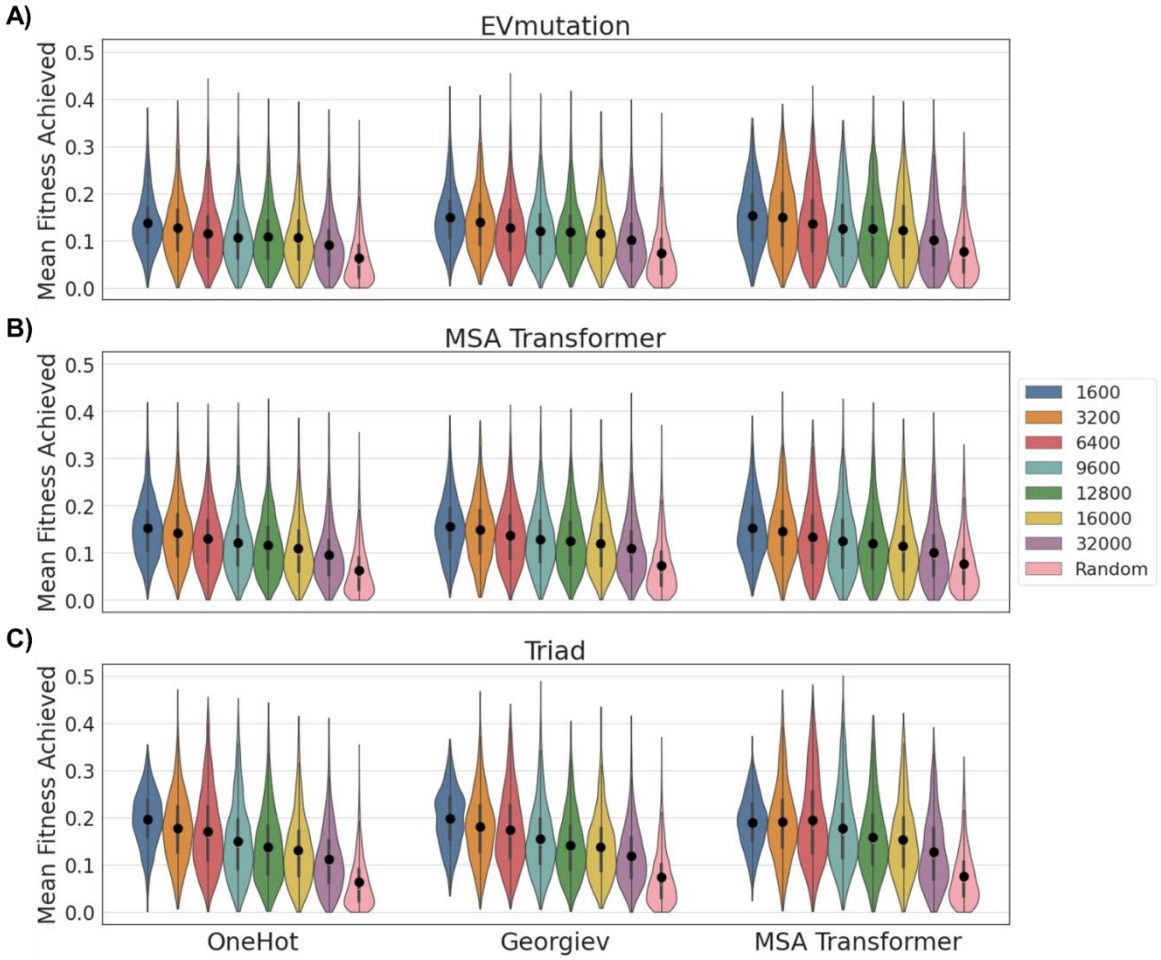


Figure A-10. Zero-shot prediction for training set design enables highly effective ftMLDE on the GB1 landscape, as measured by mean fitness achieved in simulated experiments, related to Figure 2-6. Each subplot (A–C) shows the effect of different zero-shot predictors on the mean fitness achieved in simulated ftMLDE experiments. Each violin represents data from 2000 simulated experiments where 24 variants were used for training and the top 56 predictions were tested. The major groupings of violins within each subplot correspond to different encoding strategies (one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer). The color of each violin corresponds to the zero-shot sampling threshold (i.e., the number of best-ranked variants according to a zero-shot predictor from which random samples were drawn to generate training data). Results of ftMLDE are compared to the results of standard MLDE (the three pink violins in each plot). (A) The mean fitness achieved by simulated ftMLDE when EVmutation was used as the zero-shot predictor for training set design. (B) The mean fitness achieved by simulated ftMLDE when a mask-filling protocol using the MSA Transformer was used as the zero-shot predictor for training set design. (C) The mean fitness achieved by simulated ftMLDE when predicted $\Delta\Delta G$ was used as the zero-shot predictor for training set design.

A.8 Supplemental Tables

Table A-1. The frequency with which the 2-layer 1D convolutional neural network (1D CNN) architecture appeared in the top 3 models (as ranked by cross-validation error) over 2000 rounds of simulated MLDE for each encoding type by training data size, related to Figure 2-2. The 2-layer 1D CNN was particularly effective when trained on 384 points, especially for higher-dimensional encodings. The same could not be said for the model when trained on 24 or 48 training points.

Encoding	Amino Acid Encoding Dimensionality	Percentage 2-Layer CNN in Top 3, 384 Training Points	Percentage 2-Layer CNN in Top 3, 48 Training Points	Percentage 2-Layer CNN in Top 3, 24 Training Points
Georgiev	19	14.55%	2.60%	6.30%
OneHot	20	20.55%	3.45%	5.45%
Bepler	100	18.30%	2.15%	3.75%
ResNet	256	38.05%	6.70%	6.05%
TAPE-Transformer	512	57.80%	9.35%	7.15%
MSA Transformer	768	32.80%	3.70%	4.85%
ProtBert-BFD	1024	36.85%	3.65%	3.55%
Esm1b	1280	32.80%	3.00%	4.00%
UniRep	1900	51.65%	4.55%	5.30%
LSTM	2048	62.15%	5.20%	5.20%

Table A-2. The frequency with which the 1-layer 1D convolutional neural network (1D CNN) architecture appeared in the top 3 models (as ranked by cross-validation error) over 2000 rounds of simulated MLDE for each encoding type by training data size, related to Figure 2-2. The 1-layer 1D CNN was generally not as effective as the 2-layer 1D CNN.

Encoding	Amino Acid Encoding Dimensionality	Percentage 1-Layer CNN in Top 3, 384 Training Points	Percentage 1-Layer CNN in Top 3, 48 Training Points	Percentage 1-Layer CNN in Top 3, 24 Training Points
Georgiev	19	4.20%	3.85%	6.50%
OneHot	20	9.05%	3.50%	5.90%
Bepler	100	1.90%	1.05%	2.40%
ResNet	256	2.20%	3.80%	4.25%
TAPE-Transformer	512	19.50%	9.75%	10.75%
MSA Transformer	768	11.85%	3.05%	5.45%
ProtBert-BFD	1024	6.55%	1.85%	4.90%
Esm1b	1280	11.00%	1.35%	5.90%
UniRep	1900	5.70%	1.40%	5.75%
LSTM	2048	7.95%	1.40%	6.50%

Table A-3. The frequencies with which XGBoost models with a tree base model and trained with the Tweedie regression objective achieved a greater than or equal to MLDE outcome than the same models trained with the standard regression objective, related to Figure 2-2. MLDE outcome is measured by max fitness achieved, mean fitness achieved, and NDCG. Frequencies are calculated over 2000 simulated MLDE experiments for each combination of encoding and number of training points. Instances where Tweedie gave a greater than or equal to result compared to standard regression are bolded.

Encoding	Training Points	Max, Percent Tweedie ≥ Standard	Mean, Percent Tweedie ≥ Standard	NDCG, Percent Tweedie ≥ Standard
Bepler	24	**60.85%**	**63.40%**	**67.80%**
ESM1b	24	**69.50%**	**74.85%**	**80.60%**
Georgiev	24	**63.40%**	**67.60%**	**68.60%**
LSTM	24	**69.55%**	**75.15%**	**81.15%**
MSA Transformer	24	**67.30%**	**71.70%**	**75.50%**
OneHot	24	**55.30%**	**51.80%**	42.05%
ProtBert-BFD	24	**68.65%**	**72.95%**	**77.15%**
ResNet	24	**66.95%**	**69.05%**	**72.40%**
TAPE-Transformer	24	**66.35%**	**72.00%**	**78.25%**
UniRep	24	**69.90%**	**73.95%**	**80.25%**
Bepler	48	**69.70%**	**75.95%**	**87.90%**
ESM1b	48	**77.90%**	**87.60%**	**97.60%**
Georgiev	48	**71.45%**	**75.05%**	**86.05%**
LSTM	48	**74.90%**	**84.15%**	**96.40%**
MSA Transformer	48	**73.35%**	**84.20%**	**95.40%**
OneHot	48	**65.20%**	**62.80%**	**58.05%**
ProtBert-BFD	48	**77.50%**	**87.00%**	**97.55%**
ResNet	48	**71.05%**	**77.70%**	**88.85%**
TAPE-Transformer	48	**72.75%**	**82.55%**	**94.95%**
UniRep	48	**79.30%**	**86.60%**	**97.45%**
Bepler	384	**80.50%**	**96.55%**	**100.00%**
ESM1b	384	**71.80%**	**90.40%**	**99.00%**
Georgiev	384	**72.60%**	**76.05%**	**98.80%**
LSTM	384	**71.70%**	**88.95%**	**99.90%**
MSA Transformer	384	**68.35%**	**84.30%**	**99.20%**
OneHot	384	**74.75%**	**76.15%**	**94.35%**
ProtBert-BFD	384	**75.35%**	**92.60%**	**99.50%**
ResNet	384	**81.65%**	**95.05%**	**99.80%**
TAPE-Transformer	384	**73.65%**	**91.60%**	**99.65%**
UniRep	384	**72.40%**	**95.50%**	**100.00%**

Table A-4. The frequencies with which XGBoost models with a linear base model and trained with the Tweedie regression objective achieved a greater than or equal to MLDE outcome than the same models trained with the standard regression objective, related to Figure 2-2. MLDE outcome is measured by max fitness achieved, mean fitness achieved, and NDCG. Frequencies are calculated over 2000 simulated MLDE experiments for each combination of encoding and number of training points. Instances where Tweedie gave a greater than or equal to result compared to standard regression are bolded.

Encoding	Training Points	Max, Percent Tweedie ≥ Standard	Mean, Percent Tweedie ≥ Standard	NDCG, Percent Tweedie ≥ Standard
Bepler	24	**58.85%**	**50.05%**	**73.80%**
ESM1b	24	45.40%	44.20%	**61.85%**
Georgiev	24	**87.70%**	**63.65%**	**81.35%**
LSTM	24	42.90%	42.30%	**62.60%**
MSA Transformer	24	**54.30%**	**53.55%**	**73.80%**
OneHot	24	**96.00%**	**55.10%**	**61.30%**
ProtBert-BFD	24	45.05%	43.85%	**62.90%**
ResNet	24	**67.60%**	**59.35%**	**75.95%**
TAPE-Transformer	24	49.00%	49.10%	**66.60%**
UniRep	24	46.90%	43.00%	**67.40%**
Bepler	48	**52.10%**	42.85%	**83.50%**
ESM1b	48	40.50%	41.15%	**62.80%**
Georgiev	48	**80.25%**	**67.45%**	**90.75%**
LSTM	48	30.30%	30.95%	**50.25%**
MSA Transformer	48	55.50%	**54.70%**	**80.10%**
OneHot	48	**94.45%**	**55.60%**	**67.00%**
ProtBert-BFD	48	48.35%	47.70%	**72.80%**
ResNet	48	**69.85%**	**57.20%**	**87.80%**
TAPE-Transformer	48	**57.70%**	**52.60%**	**84.45%**
UniRep	48	35.15%	31.60%	**57.95%**
Bepler	384	34.75%	17.30%	**70.60%**
ESM1b	384	46.35%	24.80%	**96.05%**
Georgiev	384	**72.10%**	42.90%	**96.75%**
LSTM	384	**64.10%**	**70.75%**	**99.90%**
MSA Transformer	384	**69.35%**	**63.15%**	**99.95%**
OneHot	384	**95.90%**	46.95%	**90.60%**
ProtBert-BFD	384	43.15%	25.10%	**99.50%**
ResNet	384	**66.00%**	**55.70%**	**92.85%**
TAPE-Transformer	384	**65.45%**	46.80%	**98.00%**
UniRep	384	49.25%	48.40%	**99.55%**

Table A-5. Expected max of the top 96 predictions, mean of the top 96 predictions, and normalized discounted cumulative gain (NDCG) for the 2000 ftMLDE simulations performed using training data designed to be enriched in fit variants, related to Figure 2-3.

Thresh	Max Fitness Achieved	Mean Fitness Achieved	NDCG
0.000	0.656	0.210	0.821
0.011	0.778	0.313	0.885
0.034	0.795	0.331	0.894
0.057	0.810	0.343	0.899
0.080	0.806	0.345	0.901

Table A-6. Effectiveness of zero shot strategies that did not rely on a mask filling protocol, related to Figure 2-4. A more positive Spearman ρ indicates a more effective prediction. The entry "Triad_FlexibleBb_ΔΔG" refers to zero-shot predictions made using predicted ΔΔG with a flexible protein backbone; the entry "Triad_FlexibleBb_RMSD" refers to zero-shot predictions made using predicted RMSD with a flexible protein backbone; the entry "Triad_FixedBb_ΔΔG" refers to zero-shot predictions made using predicted ΔΔG with a fixed protein backbone.

ZeroShotStrategy	Spearman ρ
Triad_FlexibleBb_ΔΔG	-0.02
DeepSequence	0.05
Triad_FlexibleBb_RMSD	0.06
EVmutation	0.21
Triad_FixedBb_ΔΔG	0.27

Table A-7. Effectiveness of different transformer models for zero-shot predictions of GB1 fitness using a masked token prediction protocol, related to Figure 2-4. A more positive Spearman ρ indicates a more effective prediction. All models except the MSA Transformer showed poor predictive performance when used for zero-shot prediction; the mask filling rankings provided by many models showed negative correlation with GB1 fitness, indicating a prediction that was worse than a random guess.

Model Name	Parameters (Millions)	Training Data Source	Conditional Spearman ρ	Naïve Spearman ρ
esm1_t34_670M_UR50S	670	UniRef50	-0.12	-0.09
esm1_t34_670M_UR50D	670	UniRef100 Sampled Evenly Across UniRef50 Clusters	-0.11	-0.08
esm1_t12_85M_UR50S	85	UniRef50	-0.08	-0.06
ESM1b	650	UniRef50	-0.06	-0.03
esm1_t6_43M_UR50S	43	UniRef50	-0.05	-0.03
ProtBert-BFD	420	Big Fat Database (BFD)	-0.05	0.00
ProtBert	420	UniRef100	-0.02	0.00
esm1_t34_670M_UR100	670	UniRef100	0.03	0.04
MSA Transformer	100	MSAs of Each UniRef50 Sequence Against UniClust30	0.20	0.24

Table A-8. The tunable parameters with their default values for the different neural network architectures used in MLDE.

Architecture	Parameter	Description	Default
NoHidden	dropout	Dropout value for model	0.2
OneHidden	dropout	Dropout value for model	0.2
OneHidden	size1	Size of the hidden layer as a fraction of the encoding dimensionality	0.25
TwoHidden	dropout	Dropout value for model	0.2
TwoHidden	size1	Size of the hidden layer as a fraction of the encoding dimensionality	0.25
TwoHidden	size2	Size of the second hidden layer as a fraction of the encoding dimensionality	0.0625
OneConv	dropout	Dropout value for model	0.2
OneConv	filter_choice	The width of the 1D convolutional window as a fraction of the number of positions in the combinatorial space	0.5
OneConv	n_filters1	The number of filters used in the convolution as a fraction of the encoding dimensionality of a single position	0.0625
OneConv	flatten_choice	The method of flattening post convolution	"Average"
TwoConv	dropout	Dropout value for model	0.2
TwoConv	filter_arch	The widths of the two 1D convolutional windows as a fraction of the number of positions in the combinatorial space, given as a tuple	(0.5, 0.5)
TwoConv	n_filters1	The number of filters used in the first convolution as a fraction of the encoding dimensionality of a single position	0.0625
TwoConv	n_filters2	The number of filters used in the second convolution as a fraction of the encoding dimensionality of a single position	0.007813
TwoConv	flatten_choice	The method of flattening post convolution	"Average"

Table A-9. The tunable parameters with their default values for the base models used in the XGBoost models of MLDE. “See XGBoost docs” indicates that the default values provided in XGBoost were used.

Base Model	Parameter	Default Value
Linear	lambda	1
Linear	alpha	See XGBoost docs
Tree	eta	See XGBoost docs
Tree	max_depth	See XGBoost docs
Tree	lambda	See XGBoost docs
Tree	alpha	See XGBoost docs

Table A-10. The tunable parameters with their default values for the scikit-learn models used in MLDE. "See sklearn docs" indicates that the default values provided in scikit-learn were used.

Model	Parameter	Default Value
Linear	N/A	N/A
GradientBoostingRegressor	learning_rate	See sklearn docs
GradientBoostingRegressor	n_estimators	See sklearn docs
GradientBoostingRegressor	min_samples_split	See sklearn docs
GradientBoostingRegressor	min_samples_leaf	See sklearn docs
GradientBoostingRegressor	max_depth	See sklearn docs
RandomForestRegressor	n_estimators	See sklearn docs
RandomForestRegressor	min_samples_split	See sklearn docs
RandomForestRegressor	min_samples_leaf	See sklearn docs
RandomForestRegressor	max_depth	See sklearn docs
LinearSVR	tol	See sklearn docs
LinearSVR	C	See sklearn docs
LinearSVR	dual	See sklearn docs
ARDRegression	tol	See sklearn docs
ARDRegression	alpha_1	See sklearn docs
ARDRegression	alpha_2	See sklearn docs
ARDRegression	lambda_1	See sklearn docs
ARDRegression	lambda_2	See sklearn docs
KernelRidge	alpha	See sklearn docs
KernelRidge	kernel	See sklearn docs
BayesianRidge	tol	See sklearn docs
BayesianRidge	alpha_1	See sklearn docs
BayesianRidge	alpha_2	See sklearn docs
BayesianRidge	lambda_1	See sklearn docs
BayesianRidge	lambda_2	See sklearn docs
BaggingRegressor	n_estimators	See sklearn docs
BaggingRegressor	max_samples	See sklearn docs
LassoLarsCV	max_iter	See sklearn docs
LassoLarsCV	cv	5
LassoLarsCV	max_n_alphas	See sklearn docs
DecisionTreeRegressor	max_depth	See sklearn docs
DecisionTreeRegressor	min_samples_split	See sklearn docs
DecisionTreeRegressor	min_samples_leaf	See sklearn docs
SGDRegressor	alpha	See sklearn docs
SGDRegressor	l1_ratio	See sklearn docs
SGDRegressor	tol	See sklearn docs
KNeighborsRegressor	n_neighbors	See sklearn docs
KNeighborsRegressor	weights	See sklearn docs
KNeighborsRegressor	leaf_size	See sklearn docs
KNeighborsRegressor	p	See sklearn docs
ElasticNet	l1_ratio	See sklearn docs
ElasticNet	alpha	See sklearn docs
AdaBoostRegressor	n_estimators	See sklearn docs
AdaBoostRegressor	learning_rate	See sklearn docs

Table A-11. Key resources related to Chapter 2.

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited Data		
Protein G Domain B1 (GB1) Combinatorial Fitness Landscape	(Wu et al., 2016)	DOI: 10.7554/eLife.16965
Protein G Domain B1 (GB1) Structure	Protein Data Bank (PDB)	PDB: 2GI9
Encodings, training indices, and other information needed for replicating this work; simulation results by model; summary statistics for all simulations	This Study	DOI: 10.22002/D1.1958
Additional figures too numerous to include in the appendices	This Study	https://github.com/fhalab/MLDE/SupplementalFigures; https://data.caltech.edu/badge/latestdoi/318607673
Software and Algorithms		
Triad	Protabit, Pasadena, CA, USA	https://triad.protabit.com/
Anaconda Package Manager		https://anaconda.org/; Specific package versions used in this work are given in the provided mlde.yml and mlde2.yml files on the GitHub repository associated with this work (https://github.com/fhalab/MLDE); especially important packages are listed in subsequent entries
Tasks Assessing Protein Embeddings (TAPE)	(Rao et al., 2019)	https://github.com/songlab-cal/tape-neurips2019
Evolutionary Scale Modeling (ESM)	(Rao et al., 2021; Rives et al., 2021)	https://github.com/facebookresearch/esm
ProtTrans	(Elnaggar et al., 2020)	https://github.com/agemagician/ProtTrans
Machine Learning-Assisted Directed Evolution (MLDE)	This Study	https://data.caltech.edu/badge/latestdoi/318607673
Keras Python Package		https://anaconda.org/conda-forge/keras
XGBoost Python Package	(Chen and Guestrin, 2016)	https://anaconda.org/conda-forge/xgboost
scikit-learn Python Package	(Buitinck et al., 2013)	https://anaconda.org/anaconda/scikit-learn

Appendix B

SUPPORTING MATERIAL FOR CHAPTER 4

B.1 Materials and Methods

B.1.1 Single-Site-Saturation Library Generation for TrpB

Saturation mutagenesis libraries were prepared using a modification of the "22-codon trick" described by Kille et al.[85] Primers were first designed using the templates given in Table B-12. For the forward primers, each sequence of "NNN" in these templates was replaced with "NDT," "VHG," and "TGG," resulting in a total of three degenerate primers that could then be mixed at a ratio of 12:9:1, respectively. The reverse primers were used without changes.

Primers were also designed that bind within the ampicillin resistance (AmpR) gene in pET22b(+) with sequences as given in Table B-13. These primers were designed such that, when used in combination with the site-specific primers to run a PCR, two medium-length fragments would be created with a break in the AmpR gene. For the forward site-saturation primers, a PCR was performed using the reverse AmpR primer, resulting in a fragment from ~1500–2000 bp long. For the reverse site-saturation primers, a PCR was performed using the forward AmpR primer, resulting in a fragment ~4500–5000 bp long.

Once PCRs finished, 1 µL of DpnI (NEB R0176S) was added to each of the reactions, which were then incubated at 37 °C for 1 h to digest the unmutated template plasmid. The presence of correctly sized fragments was confirmed via gel electrophoresis and each fragment was then excised from the gel and purified with the Zymoclean Gel DNA Recovery Kit (Zymo Research D4002).

Purified fragments were then assembled following the standard Gibson assembly method.[197] After 1 h at 50 °C, the reaction mixtures were desalted with a DNA Clean & Concentrator-5 kit (Zymo Research D4013) and used to transform electrocompetent E. cloni cells (Lucigen 60051-1). Libraries were spread onto solid agar selection medium consisting of Luria Broth (RPI L24040-5000.0) supplemented with 100 µg/mL carbenicillin (LB_{carb}) and incubated at

37 °C until single colonies were observed. Individual colonies were then transferred into the wells of 96-well 2-mL deep-well plates containing 300 µL of LB_{carb} to isolate monoclonal enzyme variants, with 8 wells being reserved for control conditions, giving 4-fold oversampling of the 22-codon library. These cultures were grown overnight at 37 °C, 220 rpm, and 80% humidity in an Infors Multitron HT until they reached stationary phase, at which point 100 µL from each well were mixed with an equal volume of 50% glycerol and stored at –80 °C for future use.

For protein expression, 20 µL of the remaining culture were used to inoculate 630 µL of Terrific Broth with 100 µg/mL carbenicillin (TB_{carb}). These were then grown at 37 °C, 220 rpm, and 80% humidity for 3 hours in an Infors Multitron HT, at which point they were placed on ice for 30 minutes. Following this, 50 µL of a 14 mM solution of isopropyl-β-d-thiogalactoside (IPTG; GoldBio #I2481C100) in TB_{carb} were added to each well to induce protein expression at a final concentration of 1 mM IPTG. Expression proceeded in the same Infors Multitron HT shaker as before at 22 °C, 220 rpm for roughly 18 hours. Cells were harvested via centrifugation at 4500*g* for 10 minutes, the supernatant was removed, and the plates (now containing pelleted, expressed cells) were placed at –20 °C until needed.

Once cells had been harvested, cultures for evSeq were prepared. These cultures were started from the 96-well plate glycerol stocks prepared prior to moving into the cell expression protocol; the cultures were grown overnight (~18hrs) in an Infors Multitron HT (220 rpm, 37 °C) to saturation in 96-well deep-well plates in 300 µL of LB_{carb}. These cultures were then frozen and stored at –20 °C to be used for sequencing with evSeq.

A GenBank file detailing the plasmid and primers used in this section is available on the evSeq GitHub at https://github.com/fhalab/evSeq/tree/master/genbank_files/tm9d8s.gb.

B.1.2 Sequencing TrpB Libraries with evSeq

Frozen overnight cultures (preparation detailed in the previous section) were thawed at room temperature. Libraries were then sequenced with the process described in B.2.3; the evSeq software was run using all default parameters (average_q_cutoff = 25, bp_q_cutoff = 30,

length_cutoff = 0.9, match_score = 1, mismatch_penalty = 0, gap_open_penalty = 3, gap_extension_penalty = 1, variable_thresh = 0.2, variable_count = 10) with the "return_alignments" flag thrown. The inner primers used for library preparation are in Table B-14. The barcode plates (Table B-3 – Table B-10) were paired to positions as given in Table B-15.

B.1.3 Measuring the Rate of Tryptophan Formation

Rate of tryptophan formation data was collected with the same procedure described in Rix et al. for non-heat-treated lysate preparation in the section "Indole rate measurements" with a few modifications: lysis occurred in 300 μL KPi buffer with 100 μM pyridoxal 5'-phosphate (PLP) supplemented with 1 mg/mL lysozyme, 0.02 mg/mL bovine pancreas DNase I, and 0.1x BugBuster; lysis occurred at 37 °C for 1 h.[101]

B.1.4 Four-Site-Saturation Library Generation for *Rma*NOD

Positions S28, M31, Q52, and L56 of a variant of *Rma*NOD (*Rma*NOD Y32G) were targeted for comprehensive site-saturation mutagenesis using a variant of the 22-codon trick originally described by Kille et al.[85] Due to the proximity of positions S28 and M31, it was easiest to use the same mutagenesis primers to target them; the same was done for positions Q52 and L56. Because the 22-codon trick requires three degenerate codons per position targeted, nine individual primers capturing all combinations (3 codons ^ 2 positions/per primer = 9 primers) of the degenerate codons had to be ordered for each of the two mutagenic primers. Sequences of these primers are given in Table B-16.

The primers from Table B-16 were all ordered from IDT at 100 μM. Both a "forward" and a "reverse" primer mixture were prepared by combining individual forward and reverse primers in proportion to the number of individual codons they encoded. A 10 μM forward-reverse primer mixture was then prepared by adding 10 μL of both the forward and reverse primer mixtures to 80 μL ddH_2O. Once the forward-reverse primer mixture was prepared, it was used in a PCR to build a pool of DNA fragments containing the four-site combinatorial libraries. Two fragments that captured the remainder of the *Rma*NOD gene and host plasmid

(pET22b(+)) were also produced by PCR. The primers used for these flanking fragments are given in Table B-17.

After PCR completed 1 µL DpnI (NEB R0176S) was added to each reaction. The reactions were then held at 37 °C in a thermalcycler for 1 h. The PCR fragments were then gel-extracted using a Zymoclean Gel DNA Recovery Kit (D4002).

Fragments were to eventually be assembled using Gibson assembly.[197] Because the efficiency of Gibson assembly increases with decreasing numbers of fragments, an assembly PCR was performed to combine flanking fragment 1 (see Table B-17 for details) and the variant fragment. The resultant assembled fragment was then gel-extracted, again using a Zymoclean Gel DNA Recovery Kit (D4002).

To complete construction of the library of variant plasmids, a Gibson assembly was performed to combine the assembled PCR fragment and flanking fragment 0. After Gibson assembly, the Gibson reaction was cleaned using a Monarch PCR & DNA Cleanup Kit (NEB CAT T1030L). The cleaned Gibson product was next used to transform electrocompetent E. cloni BL21 DE3. Transformed cells were spread onto solid agar selection medium consisting of Luria Broth (RPI L24040-5000.0) supplemented with 100 µg/mL ampicillin (LB_{amp}) and incubated at 37 °C until single colonies were observed.

To build the 96-well plates of *Rma*NOD variants used to demonstrate evSeq, 400 µL LB + 100 µg/mL ampicillin were first added to each well of 5x 96-well deepwell plates. Colonies from the agar plates grown overnight were then picked into the wells of the deepwell plates. The plates were placed in an Infors Multitron HT at 240 rpm, 37 °C for ~16 h. To glycerol stock the now-stationary-phase culture, 100 µL overnight culture were added to 100 µL 50% glycerol before being stored at -80 °C until its use in evSeq library preparation.

A GenBank file detailing the plasmid and primers used in this section is available on the evSeq GitHub (https://github.com/fhalab/evSeq/tree/master/genbank_files/rmanod_y32g.gb).

B.1.5 Sequencing *Rma*NOD Libraries with evSeq

To begin preparation of culture for evSeq with the *Rma*NOD variants, cultures in 96-well deep-well plates (with 300 µL of LB_{carb}) were started from the 96-well plate glycerol stocks prepared in the previous section. The plates were placed in an Infors Multitron HT at 240 rpm; the cultures were grown overnight (~18hrs) before being frozen and stored at –20 °C.

To start the evSeq protocol, frozen overnight cultures were thawed in a room temperature water bath. Libraries were then sequenced with the process described in B.2.3; the evSeq software was run using the same parameters as for the TrpB data analysis (see Section B.1.2). The inner primers used for evSeq library preparation are given in Table B-18. The barcode plates (Table B-3 – Table B-10) were paired to positions as given in Table B-19.

B.1.6 Oligo Design

B.1.6.1 Inner Primer Design

The inner primers of evSeq are specific to the region of interest. Each region of interest is captured by both a forward and reverse primer. These primers have the below general layout:

F: 5’ – CACCCAAGACCACTCTCCGG**XXXXXXX**… – 3’

R: 5’ – CGGTGTGCGAAGTAGGTGC**XXXXXXXX**… – 3’

The 5’ region is a universal adapter to which outer primers bind (see Section B.2.2) while the 3’ region (denoted by “**X**” in the primers above) is specific to the region of interest. Note that the length of the variable 3’ region will vary depending on the target gene (this is indicated by the ellipses at the end of the poly-**X** region). Note that there is no need for the two primers in the pair to be equal length—I show them as such to highlight the fact that the forward universal adapter is one base longer than the reverse universal adapter. Detailed instructions for effective primer construction are provided on the evSeq wiki (https://fhalab.github.io/evSeq/1-lib_prep.html#inner-primer-design).

B.1.6.2 Outer Primer Design

The barcode (outer) primers used in evSeq all follow the below layout:

F: 5’ – TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG**XXXXXXX**CACCCAAGACCACTCTCCGG – 3’

R: 5’ – GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG**XXXXXXX**CGGTGTGCGAAGTAGGTGC – 3’

Each of these primers consists of (1) a 5’ sequence matching the Illumina Nextera transposase adapters, (2) a central unique 7-nucleotide barcode (Table B-1), and (3) a 3’ universal seed that matches the 5’ adapter of the inner primers (see Section B.1.6.1). Note that only Illumina indices compatible with the Nextera transposase adapters can be used with the provided outer primer designs; other indexing systems would require different adapters. The full set of outer primers used in this study can be found in Table B-2; they can be ordered from IDT by following the instructions provided in B.2.1.

B.1.6.3 Barcode Design

evSeq uses 192 unique 7-nucleotide barcodes (Table B-1). The barcodes were designed to satisfy the below criteria:

1. All barcodes must have GC-content of 40–60%.

2. All barcodes must be at least 3 substitutions apart. This is to prevent misassignment of reads due to sequencing errors of the barcodes.

3. No barcode can have 3 of the same bases in a row. This is to reduce sequencing errors.

4. No barcode can be a sub-sequence of the Nextera transposase adapters or their reverse complements (see below). This is to avoid interference with downstream Illumina chemistry.

5. No barcode can be a sub-sequence of the Illumina p5 and p7 flow cell-binding sequences or their reverse complements (see below for sequences). Again, this is to avoid interference with downstream Illumina chemistry.

The Nextera transposase adapter sequences are below:

5’ – TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG – 3'

5’ – GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG – 3'

The p5 and p7 flow cell-binding sequences are below:

p5: 5’ – AATGATACGGCGACCACCGAGATCTACAC – 3'

p7: 5’ – CAAGCAGAAGACGGCATACGAGAT – 3’

B.2 Protocols

B.2.1 Ordering Barcode Primers from IDT

I provide a pre-filled IDT order form for all evSeq primers on the evSeq GitHub repository (https://github.com/fhalab/evSeq/tree/master/lib_prep_tools/IdtOrderForm.xlsx). This order form can be used to order evSeq primers in the 96-well plate layout needed to prepare the evSeq barcode primer mixes (see Section B.2.2). To order evSeq primers:

1. Navigate to the IDT DNA oligo ordering page: https://www.idtdna.com/pages/products/custom-dna-rna/dna-oligos/custom-dna-oligos.

2. Under “Ordering,” select “Plates.”

3. From the “Single-stranded DNA” table, select the amount (in nanomoles) of oligo you wish to order (denoted in the “Product” column) by clicking “Order” under the “96 Well” column. For the work described in Chapter 4, 25 nmole oligos were ordered.

4. On the next page, click “UPLOAD PLATE(S).” Using the pop-up that results, upload the “IdtOrderForm.xls” provided on the evSeq GitHub repository. The pop-up should recognize two plates—one called “FBC” and the other called “RBC”—each

consisting of 96 wells. Click “ADD PLATES” followed by “CLOSE THIS WINDOW” to close the window.

5. For the “FBC” plate, click “Plate Specifications.” Confirm that the below specifications are set as follows:

    a. Purification: Standard Desalting

    b. Plate Type: Deep Well

    c. **Ship Option: Wet**

    d. Buffer: IDTE 8.0 pH

    e. Normalization Type: Full Yield

    f. **Concentration: 100 µM**

Note that the bolded specifications are different from default. **While not strictly required, it is strongly recommended that primers be ordered *wet* at 100 µM; reconstituting plates of dry primers to 100 µM can be very time-consuming without robotic support.**

6. Once specifications are correctly set for the “FBC” plate, click “APPLY SETTINGS TO ALL PLATES” at the bottom of the specifications pop-up, followed by “YES” on the window that follows. Quickly check to make sure that the same settings as recommended in step 5 were applied to “RBC” by clicking on the “RBC” “Plate Specifications” option.

7. Add the primers to your order by clicking “ADD TO ORDER,” then follow standard IDT procedures for purchasing.

B.2.2 Preparation of evSeq Barcode Primer Mixes

There are 96 unique forward and 96 unique reverse outer primers (Table B-2), corresponding to 96 unique forward and 96 unique reverse barcodes (Table B-1). The forward and reverse outer primers were ordered following the procedure given above in Section B.2.1.

Each well sequenced in evSeq is encoded by a different combination of forward and reverse barcode. Different primers from the forward and reverse outer primer plates can be mixed together to associate a barcode combination with a specific well in a specific plate. Because the same outer primers can be used regardless of inner primer, it is convenient to keep plates of barcode combinations on hand. Plates of outer primer combinations (hereafter also referred to as "barcode plates") can be stored for long periods of time.

Throughout the development of evSeq, the same 8 barcode plates were used (consisting of 768 different combinations of forward and reverse outer primers) to encode plate and well locations. Barcode plates are named DI01–DI08, where "DI" stands for "dual-indexed." The exact barcode combinations used by evSeq are given in Table B-3 – Table B-10; these combinations can also be found in the "index_map.csv" file on the evSeq GitHub: (https://github.com/fhalab/evSeq/tree/master/evSeq/util/index_map.csv). By default, the evSeq software assumes the barcode plates used for library preparation are laid out in the order given in the "index_map.csv" file. To build the barcode plates depicted in Table B-3 – Table B-10, the below procedure was followed:

1. 10-fold dilutions of each of the forward and reverse outer primer plates ordered from IDT were prepared by adding 10 µL of each primer stock to 90 µL ddH2O, keeping the well layout constant. Dilutions were performed in fully-skirted PCR plates (Bio-Rad HSP9601). The plates from IDT had a starting concentration of 100 µM, so the final concentration of these two diluted plates was 10 µM.

2. To 8 fully-skirted PCR plates, 80 µL ddH2O were added, followed by 10 µL diluted (10 µM) forward barcode plate. The well layout was kept constant for the forward barcode primers.

3. To the each of the 8 plates, 10 μL of diluted (10 μM) reverse barcode plate were added, shifting the well layout down by 1 row per plate. For instance, row A of the reverse plate went into row A of the first barcode plate, row B of the second barcode plate, row C of the third barcode plate, and so on; row H of the reverse plate went into row H of the first barcode plate, row A of the second barcode plate, row B of the third barcode plate, and so on.

4. When not in use, the 10-fold dilutions prepared in step 1 were stored at –20 °C, while the barcode plates (each well of which had a combination of a specific forward and reverse primer at a final concentration of 1 μM) were stored at 4 °C. Both the 10 μM stock plates and 1 μM barcode plates can be stored for long periods of time—I have noticed no drop in effectiveness even after years of storage.

B.2.3 evSeq Library Preparation/Data Analysis Protocol

The evSeq library preparation protocol was designed to be as cost-effective as possible. The quantities used in the below protocol were chosen to fit within the constraints of the resources available to the Arnold Lab (these are the quantities used for all evSeq experiments performed in Chapter 4). However, with automation support (e.g., liquid handling robots) and higher-capacity molecular biology equipment, the entire protocol could be scaled down to lower quantities, further improving cost-effectiveness.

The list of steps below can be followed to prepare an evSeq library for sequencing using the outer primers described in Section B.2.2. Note that when first using a new set of inner primers, it is recommended to complete the below protocol for a few wells as a test before deploying them for plate-scale reactions.

The library preparation protocol can be completed with the below steps. Note that provided part numbers are for the materials/reagents used while developing this protocol—the same components from other providers will almost certainly work as well. This protocol is also provided on the evSeq wiki (https://fhalab.github.io/evSeq/1-lib_prep.html#pcr-protocol).

1. Prepare a PCR master mix for the number of wells to be sequenced according to the below table. Note that an excel calculator is provided on the evSeq GitHub repository for easy calculation of master mix volumes based on the number of plates to be sequenced (https://github.com/fhalab/evSeq/tree/master/lib_prep_tools/MastermixCalculator.xlsx).

Component	**Amount per 10 µL rxn (µL)**
Thermopol Buffer (NEB B9004S)	1.00
10 mM dNTPs (NEB N0447)	0.20
Taq Polymerase (NEB M0267)	0.05
ddH_2O	5.33
Mol-Bio Grade DMSO (MP 194819)	0.40
Inner Primer Mix (10 µM)	0.02

   a. Note that the above table assumes that each evSeq PCR reaction will be 10 µL—if scaling down, adjust volumes accordingly.

   b. Note that the above table also assumes the same set of inner primers is used to prepare all plates. If this is not the case, a separate master mix will need to be prepared for each set of inner primers.

   c. The Inner Primer Mix (10 µM) is a combination of forward and reverse inner primers at a final concentration of 10 µM each in diH2O (this can be prepared, e.g., by adding 10 µL of 100 µM forward inner primer and 10 µL of 100 µM reverse inner primer to 80 µL diH2O).

2. Add 7 µL of master mix to each well of as many half-skirted PCR plates (USA Scientific 1402-9700) as will be sequenced. These are referred to as “PCR plates” in the remainder of this protocol.

3. Stamp 1 µL of overnight culture from each plate to be sequenced into the PCR plates.

a. “Stamp” means “apply to all wells, keeping the plate layout consistent.” For example, 1 µL of culture from library 01 F02 is moved to PCR plate 01 F02, 1 µL of culture from library 02 C07 is moved to PCR plate 02 C07, etc.

b. Note that both fresh culture and previously frozen culture (thawed before use as template) will work here. No modifications need to be made to the protocol.

4. Complete the stage 1 PCR using the below thermalcycler conditions. This PCR amplifies the fragment of interest from the template DNA contained in the cell culture.

Step	Temperature (°C)	Time
1	95	5 min
2	95	20 s
3	TD 63-> 54	20 s
4	68	30 s
5	Return to 2, 9 x	
6	4	Hold

a. "TD” above stands for “touchdown.” A touchdown step decrements the temperature by 1 °C each cycle. The touchdown in the above PCR starts at 63 °C and drops to 54 °C by the end.

b. Note that the extension step (step 4) is long enough to amplify a 500 bp fragment. Longer fragments will need a longer extension time. Note, however, that you may see reduced sequencing efficiency with fragments that are too large.

c. While developing this protocol, the below thermal cycler models were used:

i. Eppendorf Mastercycler ep Gradient S Thermal Cycler, Model 5345 with 96-well universal block

ii. Eppendorf Mastercycler pro S vapo.protect

iii. Eppendorf Mastercycler X50s 96-well silver block thermal cycler

5. Once PCR has completed, stamp 2 µL of 1 µM barcode primer mix from the barcode plates into the PCR plates (see Section B.2.2 for details on preparation of barcode plates). **Record which barcode plate was stamped into which PCR plate.**

6. Perform the second step PCR using the below conditions:

Step	Temperature (°C)	Time
1	95	20s
2	68	50 s
3	Return to 1, 24 x	
4	68	5 min
5	4	Hold

   a. Again, longer fragments may need a longer extension time.

7. While the second PCR runs, prepare a 2% agarose gel with SYBR gold added (Thermo Fisher Scientific, S11494).

8. Once the second PCR has completed, for each plate, pool 5 µL of each reaction into 100 mM EDTA to a final concentration of 20 mM EDTA—this step quenches the reactions. Pooling will leave you with as many tubes as you have plates, each containing ~600 µL [96 rxns/plate × (5 µL per rxn + 1.25 µL 100mM EDTA per reaction)].

   a. Note: The most efficient way to do the pooling varies depending on the equipment available. The Arnold Lab relies on 12-channel multichannel pipets for this step, and so will accomplish pooling by (1) adding 10 µL 100 mM EDTA to each well in a single row of a fresh PCR plate, (2) transferring 5 µL reaction from each row in the plate-to-be-pooled into the single row of EDTA, and (3) transferring 40 µL from each well in the single row of pooled

reactions using a single-channel pipet (leaving 10 µL dead volume in each well) to a microcentrifuge tube. An alternate strategy might be, for instance, adding 120 µL 100 mM EDTA to a trough, then pipetting 5 µL of all reactions from a plate into this trough. Whatever strategy is taken, what is important in pooling is that the ratios of the reactions in the pool remain equal—sacrificing some reaction as dead volume is perfectly acceptable to achieve equal mixing in this step.

9. For each tube made in step 8, take 100 µL of pooled reaction and add it to 20 µL 6x loading dye (NEB B7025S) in a microcentrifuge tube. **It is critical that the loading dye does not contain SDS.** At this point, the remaining pooled reaction from step 8 can be stored at –20 °C for future use (i.e., if the later steps of this protocol ever need to be redone).

    a. Note that most of the pooled reaction is not moved into later steps with this protocol. Again, if relevant automation and molecular biology equipment is available, reactions can be scaled down below 10 µL, reducing wasted reaction. Current reaction sizes are set to minimize pipetting error.

10. Load the contents of each tube made in step 9 into the agarose gel prepared in step 7. The contents of each tube should be kept separate (i.e., loaded into different lanes in the gel). Load a ladder (I typically use 100 bp ladder from NEB, N3231S) in the flanking lanes.

11. Run the agarose gel at 130 V until the bands have sufficiently migrated. Often, you will see two bands: the lower band is usually primer dimer and the upper is the target. Reference the ladder to identify your product, remembering that the two-step PCR adds 120 bp of additional length (from the universal adapter, barcode, and transposase adapters) onto the gene fragment of interest.

12. Gel-extract the target bands from the agarose gel, again keeping bands from different plates separate. I typically use Zymoclean Gel DNA Recovery Kit (Zymo Research, D4001) for this step. Elution should be performed at a low volume—I typically elute in 10 μL of ddH2O.

13. After gel extraction, combine the gel-extracted pools from each plate in equimolar concentrations. A calculator on the evSeq GitHub repository is provided that can be used to normalize *equal-length* fragments to a pre-specified concentration (https://github.com/fhalab/evSeq/tree/master/lib_prep_tools/LibDilCalculator.xlsx).

    a. Note that the quantification here need not be extremely robust. For all results presented in Chapter 4, this step was performed using DNA concentrations output by a GE NanoVue Plus.

    b. Tip: It is generally not advised to pool amplicons drastically different in length. Shorter fragments are preferentially sequenced in NGS, and so the shorter amplicon will dominate the number of reads. Separate submissions should be made for libraries with very different lengths.

14. After the previous step, you should have a single tube of cleaned, normalized DNA consisting of all amplicons from all plates to be pooled. This DNA will be submitted to your sequencing provider for inclusion in a multiplexed sequencing run. You should work with your sequencing provider to ensure that all requirements are met to slot into their pipeline. For instance, this protocol assumes that the sequencing provider can add Nextera-compatible Illumina indices and flow-cell-binding sequences via PCR—it should be confirmed that your sequencing provider can do this before submitting your sample.

    a. Note: Throughout the development of evSeq, I used the “Customized PCR Amplicon Sequencing” services of Laragen Inc. (http://www.laragen.com/laragen_nextgen.php).

b. Also note that, depending on your sequencing provider, it may be possible (or even necessary) to add the Illumina indices yourself. Again, you should work with your provider to determine the best course of action for submitting evSeq libraries. Adding indices simply requires one final PCR on the pooled evSeq library.

15. Once sequencing is complete, your sequencing provider should return two fastq (or fastq.gz) files to you. One will contain the forward reads for your pooled samples and the other will contain the reverse reads—both files are needed by the evSeq software for processing.

16. Using the files returned in step 15, run the evSeq software to process results and assign variants to their original wells. Detailed instructions on how to use the evSeq software and interpret its outputs are provided on the evSeq Wiki https://fhalab.github.io/evSeq/4-usage.html

B.3 Supplemental Figures

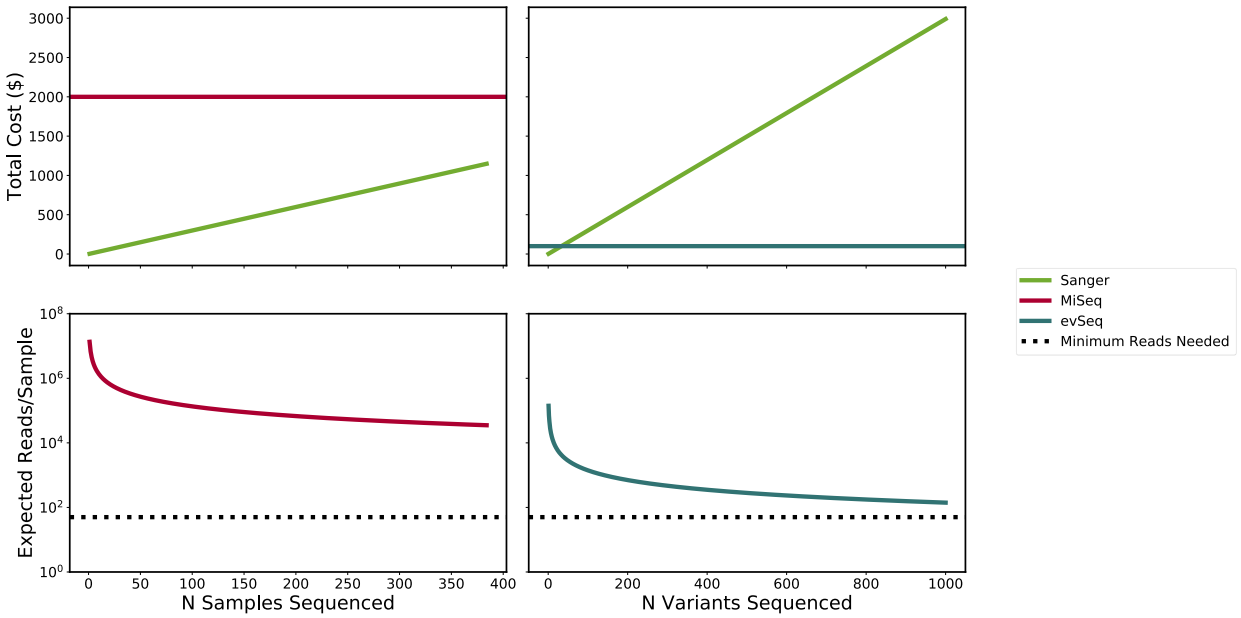


Figure B-1. Comparison of the tradeoff between sequencing depth and cost for Sanger sequencing (green), a multiplexed MiSeq run (red), and an evSeq library (blue). The top row gives the total cost for sequencing a given number of variants; the bottom row gives the expected number of reads per variant for sequencing a given number of variants. Note that the x-axes for the left and right columns are different. The limit on the x-axis for the left column is set to reflect what is typically the maximum level of multiplexed NGS available (384 samples) when outsourcing sequencing. To be consistent with the language used throughout Chapter 4, the x-axis labels refer to elements run in a multiplexed NGS run as "samples" and elements contained in an evSeq library as "variants." I assume that the elements sequenced in these examples are derived from protein mutant libraries amenable to sequencing by evSeq (i.e., the sequenced elements are targeted amplicons). Top Row: We see that both multiplexed NGS on a commercial MiSeq run and evSeq have constant cost with an increasing number of elements sequenced; Sanger, in contrast, scales linearly with the number of elements sequenced. Many elements (669 with the cost estimates used to make this figure) need to be added to a multiplexed MiSeq run before it becomes more cost-effective than Sanger. Even though research groups may frequently meet or exceed 669 variants in a standard protein engineering experiment, the flat cost of $2000 is far too high to justify regular sequencing of every variant. Many fewer variants (34) need to be added to an evSeq run before it becomes cost-effective over Sanger. A flat cost of ~$100 is justifiable for regularly sequencing all variants. Bottom Row: NGS technologies trade off sequencing depth for cost effectiveness. Notably, the per-sample sequencing depth achieved by commercially available multiplexed runs is much higher than what is needed for reliable sequencing. evSeq, in contrast, more efficiently spreads reads, keeping the expected number of reads closer to, yet still above the minimum needed for effective sequencing. Notes on Figure Generation: Cost of a single MiSeq run ($2000) is based on an estimate provided by

Laragen Inc. Cost of a single Sanger sequencing run ($2.99) is based on a quote from MCLAB for sequencing a single 96-well plate. The number of expected reads from a MiSeq run (13.5 million) is based on estimates provided by Illumina for a MiSeq Reagent Kit v2 (note that almost double the number of reads can be achieved using a v3 kit—I used v2 here to be conservative with my estimates for NGS/evSeq). The number of expected reads for a variant sampled with evSeq assumes the evSeq library was sequenced as 1 of 96 samples on a multiplexed sequencing run using a MiSeq Reagent Kit v2. The cost of a single evSeq run is based on an estimate provided by Laragen for a single sample in a multiplexed sequencing run using a PE150 kit.

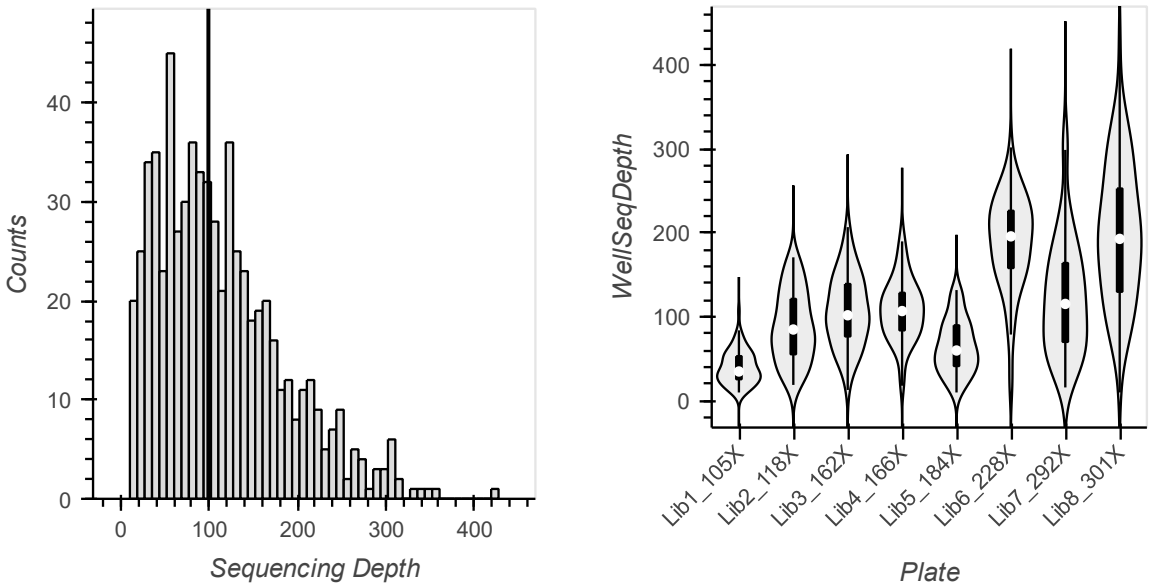


Figure B-2. Sequencing depths for the *Tm*TrpB9D8* evSeq libraries. Left: A histogram of sequencing depths for each *Tm*TrpB9D8* variant contained in the full evSeq library. The vertical black line gives the median. Right: Violin plots showing the distribution of read depths over the wells in each sequenced plate. Variability between plates likely indicates inaccurate quantification of pooled plates prior to final assembly of the evSeq library. Notable, libraries 1-5 use different evSeq primers than libraries 6-8.

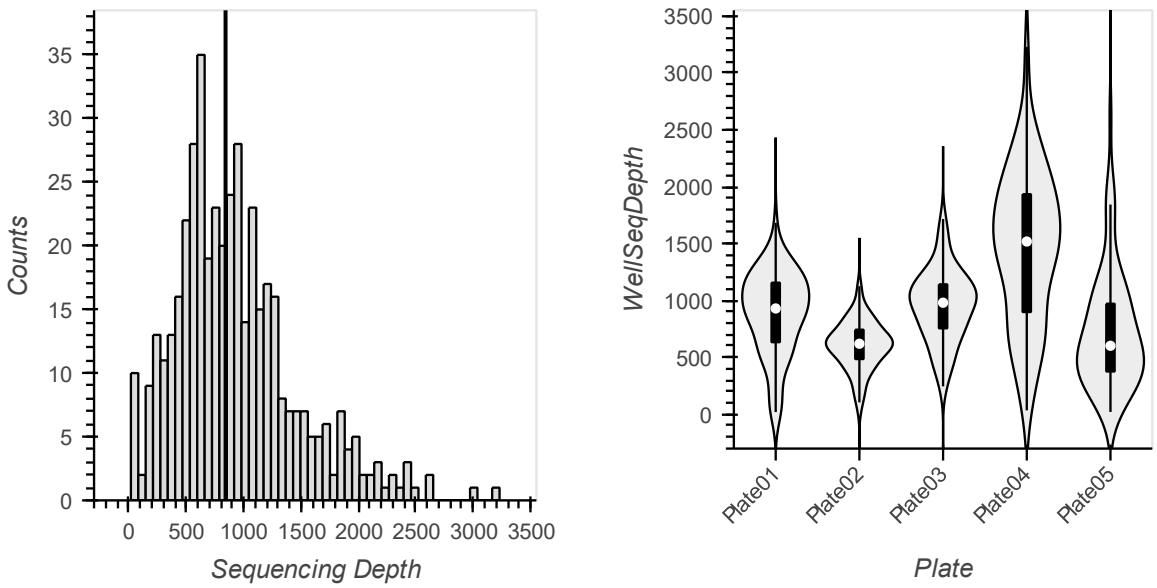


Figure B-3. Sequencing depths for the *Rma*NOD evSeq libraries. Left: A histogram of sequencing depths for each *Rma*NOD variant contained in the full evSeq library. The vertical black line gives the median. Right: Violin plots showing the distribution of read depths over the wells in each sequenced plate. Variability between plates likely indicates inaccurate quantification of pooled plates prior to final assembly of the evSeq library.

B.4 Barcode and Outer Primer Sequences

Table B-1. evSeq barcode sequences used in Chapter 4. The "Plate" and "Well" columns give the location of these sequences in the IDT order form provided on the evSeq GitHub repository (see Ordering Barcode Primers from IDT and Barcode Design, above). Note that barcode sequences can also be found in the "index_map.csv" file found on the evSeq GitHub repository (https://github.com/fhalab/evSeq/tree/master/evSeq/util/index_map.csv); this csv file also gives the combinations of barcodes used to define the dual indexing (DI) plates.

Plate	Well	Barcode
FBC	A01	GATCATG
FBC	A02	TACATGG
FBC	A03	AAGCACC
FBC	A04	TGGCTCA
FBC	A05	CTTGCTC
FBC	A06	GAAGCGT
FBC	A07	TCTCCAT
FBC	A08	TTGAAGG
FBC	A09	GAATGTC
FBC	A10	ATCTCCA
FBC	A11	GCGTTAT
FBC	A12	TGCACCA
FBC	B01	TGCCTAT
FBC	B02	AGGAATC
FBC	B03	TCCACTG
FBC	B04	TTGTACC
FBC	B05	TTCGAGT
FBC	B06	CTTCAGC
FBC	B07	CAGTGCA
FBC	B08	TGCTGTC
FBC	B09	CGCCATT
FBC	B10	GCCATGA
FBC	B11	CACAACG
FBC	B12	CTTCGCT
FBC	C01	TCGTGAA
FBC	C02	TTATCGG
FBC	C03	AGACCAT
FBC	C04	ACATGAG
FBC	C05	ACGTACT
FBC	C06	CACCTCA
FBC	C07	GTTGGAG
FBC	C08	TGTTCTG

FBC	C09	CTTACGT
FBC	C10	GAGGTTG
FBC	C11	ATGGACA
FBC	C12	ACACTGA
FBC	D01	ATCTGTG
FBC	D02	AATGTGC
FBC	D03	GAGTTGA
FBC	D04	TTCTCAC
FBC	D05	TGAAGCG
FBC	D06	GCTACAA
FBC	D07	AGAGAAC
FBC	D08	CAGAGTG
FBC	D09	TTCCGAA
FBC	D10	GTACGAC
FBC	D11	ACTCTTG
FBC	D12	CCAACCA
FBC	E01	CTCTAGA
FBC	E02	AATCGGA
FBC	E03	CGTCCTA
FBC	E04	GGAATGT
FBC	E05	TCCAAGC
FBC	E06	GCACCTA
FBC	E07	TTGCGTT
FBC	E08	CAGGATT
FBC	E09	CTGCATA
FBC	E10	CGTTGAG
FBC	E11	TGCTACT
FBC	E12	GTGATCC
FBC	F01	GCATGGT
FBC	F02	GTCGTTA
FBC	F03	CCTGACA
FBC	F04	AGTGTAG
FBC	F05	CGAGCAA
FBC	F06	CTACTCC
FBC	F07	GATGCCA
FBC	F08	GACCGAT
FBC	F09	ACGTTGG
FBC	F10	ATGAGCG
FBC	F11	TACTCCG
FBC	F12	GATTCAC
FBC	G01	ATGACGC
FBC	G02	GGTTGTT

FBC	G03	GTACTTG
FBC	G04	TAGCAAG
FBC	G05	CTGCCAT
FBC	G06	GAGAACA
FBC	G07	GTATAGC
FBC	G08	TGATGGA
FBC	G09	GGCAGTA
FBC	G10	GAAGAAG
FBC	G11	AGCGGTT
FBC	G12	TAAGGCC
FBC	H01	AACCTGT
FBC	H02	AGTACAC
FBC	H03	CTCGTAG
FBC	H04	CTAGGTG
FBC	H05	CGATACC
FBC	H06	TCGGCTA
FBC	H07	CGGTTGT
FBC	H08	ATTGCCT
FBC	H09	CATTCGA
FBC	H10	GCACAAT
FBC	H11	GCAGTAA
FBC	H12	CCTAATC
RBC	A01	GAACTGC
RBC	A02	ACCAGGT
RBC	A03	TCTAGAG
RBC	A04	CACACAA
RBC	A05	GTGGAAC
RBC	A06	ATATGCC
RBC	A07	GGTCTGA
RBC	A08	GTGAGAT
RBC	A09	TTGGCAG
RBC	A10	ATGCCTG
RBC	A11	TCCGAAG
RBC	A12	GGCTTAC
RBC	B01	AGTTGGC
RBC	B02	AACGATG
RBC	B03	ACTACCG
RBC	B04	GGTGTCT
RBC	B05	CCAGCTT
RBC	B06	TTAGACG
RBC	B07	ACCATAC
RBC	B08	GACGACT

RBC	B09	GTCACCT
RBC	B10	CGTGATG
RBC	B11	GCTTCCT
RBC	B12	TAGACGT
RBC	C01	CGGACTT
RBC	C02	ACCGGAA
RBC	C03	CCGAAGT
RBC	C04	TCACGCA
RBC	C05	ATCCTCG
RBC	C06	CGAATAG
RBC	C07	TATCCGG
RBC	C08	AGCAAGA
RBC	C09	TGTCGAC
RBC	C10	TTCCATG
RBC	C11	GCAATCG
RBC	C12	TGAGTGG
RBC	D01	TAGGAGA
RBC	D02	AGTCAGT
RBC	D03	GTGCTGT
RBC	D04	CAACAAC
RBC	D05	AATAGCC
RBC	D06	TCTGTGA
RBC	D07	TGTGGTA
RBC	D08	GCGTATG
RBC	D09	AGTTACG
RBC	D10	TTCCTGC
RBC	D11	TATGTCG
RBC	D12	GGAGAGA
RBC	E01	CCTTAGG
RBC	E02	TGTATCC
RBC	E03	CAACCTG
RBC	E04	CTGATGA
RBC	E05	AAGACAG
RBC	E06	AGCTCGT
RBC	E07	GATTGCG
RBC	E08	TCCTTCA
RBC	E09	TCACAGG
RBC	E10	AGAGCTG
RBC	E11	CCTCTGT
RBC	E12	CCTCGAA
RBC	F01	GTGTCTC
RBC	F02	ATTGAGG

RBC	F03	GACAATC
RBC	F04	CACTTGC
RBC	F05	TGAACGC
RBC	F06	CGTAGCA
RBC	F07	AGGTTCC
RBC	F08	GTACACA
RBC	F09	GATAGGT
RBC	F10	TAGCCTC
RBC	F11	TTCAGCC
RBC	F12	GGATTCA
RBC	G01	TGAGCCT
RBC	G02	AACGCGA
RBC	G03	TCATTGC
RBC	G04	AGCATCT
RBC	G05	TTGGTCT
RBC	G06	CAAGGAT
RBC	G07	AGACGTC
RBC	G08	AGGTCAA
RBC	G09	ATGCTAC
RBC	G10	CTCTGAT
RBC	G11	TCAAGTC
RBC	G12	TCGAGCT
RBC	H01	ACAGTCT
RBC	H02	CAGATAC
RBC	H03	TACGTTC
RBC	H04	ACGGTTC
RBC	H05	CATCGTC
RBC	H06	TACGCAT
RBC	H07	CTTAGAC
RBC	H08	AACTGAC
RBC	H09	ACTTGCA
RBC	H10	ACGCGAT
RBC	H11	TCGACAC
RBC	H12	ACTCAAC

Table B-2. Full-length evSeq barcode (outer) primer sequences used in Chapter 4. The "Plate" and "Well" columns give the location of these sequences in the IDT order form provided on the evSeq GitHub repository (see Ordering Barcode Primers from IDT and Preparation of evSeq Barcode Primer Mixes, above).

Plate	Well	Sequence
FBC	A01	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGATCATGCACCCAAGACCACTCTCCGG
FBC	A02	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTACATGGCACCCAAGACCACTCTCCGG
FBC	A03	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAAGCACCCACCCAAGACCACTCTCCGG
FBC	A04	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTGGCTCACACCCAAGACCACTCTCCGG
FBC	A05	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTTGCTCCACCCAAGACCACTCTCCGG
FBC	A06	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGAAGCGTCACCCAAGACCACTCTCCGG
FBC	A07	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTCTCCATCACCCAAGACCACTCTCCGG
FBC	A08	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTTGAAGGCACCCAAGACCACTCTCCGG
FBC	A09	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGAATGTCCACCCAAGACCACTCTCCGG
FBC	A10	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGATCTCCACACCCAAGACCACTCTCCGG
FBC	A11	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGCGTTATCACCCAAGACCACTCTCCGG
FBC	A12	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTGCACCACACCCAAGACCACTCTCCGG
FBC	B01	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTGCCTATCACCCAAGACCACTCTCCGG
FBC	B02	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAGGAATCCACCCAAGACCACTCTCCGG
FBC	B03	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTCCACTGCACCCAAGACCACTCTCCGG
FBC	B04	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTTGTACCCACCCAAGACCACTCTCCGG
FBC	B05	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTTCGAGTCACCCAAGACCACTCTCCGG
FBC	B06	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTTCAGCCACCCAAGACCACTCTCCGG
FBC	B07	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCAGTGCACACCCAAGACCACTCTCCGG
FBC	B08	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTGCTGTCCACCCAAGACCACTCTCCGG
FBC	B09	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCGCCATTCACCCAAGACCACTCTCCGG
FBC	B10	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGCCATGACACCCAAGACCACTCTCCGG
FBC	B11	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCACAACGCACCCAAGACCACTCTCCGG
FBC	B12	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTTCGCTCACCCAAGACCACTCTCCGG
FBC	C01	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTCGTGAACACCCAAGACCACTCTCCGG
FBC	C02	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTTATCGGCACCCAAGACCACTCTCCGG
FBC	C03	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAGACCATCACCCAAGACCACTCTCCGG
FBC	C04	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGACATGAGCACCCAAGACCACTCTCCGG
FBC	C05	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGACGTACTCACCCAAGACCACTCTCCGG
FBC	C06	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCACCTCACACCCAAGACCACTCTCCGG
FBC	C07	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGTTGGAGCACCCAAGACCACTCTCCGG
FBC	C08	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTGTTCTGCACCCAAGACCACTCTCCGG
FBC	C09	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTTACGTCACCCAAGACCACTCTCCGG
FBC	C10	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGAGGTTGCACCCAAGACCACTCTCCGG
FBC	C11	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGATGGACACACCCAAGACCACTCTCCGG

FBC	C12	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGACACTGACACCCAAGACCACTCTCCGG
FBC	D01	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGATCTGTGCACCCAAGACCACTCTCCGG
FBC	D02	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAATGTGCCACCCAAGACCACTCTCCGG
FBC	D03	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGAGTTGACACCCAAGACCACTCTCCGG
FBC	D04	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTTCTCACCACCCAAGACCACTCTCCGG
FBC	D05	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTGAAGCGCACCCAAGACCACTCTCCGG
FBC	D06	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGCTACAACACCCAAGACCACTCTCCGG
FBC	D07	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAGAGAACCACCCAAGACCACTCTCCGG
FBC	D08	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCAGAGTGCACCCAAGACCACTCTCCGG
FBC	D09	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTTCCGAACACCCAAGACCACTCTCCGG
FBC	D10	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGTACGACCACCCAAGACCACTCTCCGG
FBC	D11	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGACTCTTGCACCCAAGACCACTCTCCGG
FBC	D12	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCCAACCACACCCAAGACCACTCTCCGG
FBC	E01	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTCTAGACACCCAAGACCACTCTCCGG
FBC	E02	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAATCGGACACCCAAGACCACTCTCCGG
FBC	E03	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCGTCCTACACCCAAGACCACTCTCCGG
FBC	E04	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGGAATGTCACCCAAGACCACTCTCCGG
FBC	E05	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTCCAAGCCACCCAAGACCACTCTCCGG
FBC	E06	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGCACCTACACCCAAGACCACTCTCCGG
FBC	E07	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTTGCGTTCACCCAAGACCACTCTCCGG
FBC	E08	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCAGGATTCACCCAAGACCACTCTCCGG
FBC	E09	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTGCATACACCCAAGACCACTCTCCGG
FBC	E10	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCGTTGAGCACCCAAGACCACTCTCCGG
FBC	E11	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTGCTACTCACCCAAGACCACTCTCCGG
FBC	E12	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGTGATCCCACCCAAGACCACTCTCCGG
FBC	F01	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGCATGGTCACCCAAGACCACTCTCCGG
FBC	F02	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGTCGTTACACCCAAGACCACTCTCCGG
FBC	F03	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCCTGACACACCCAAGACCACTCTCCGG
FBC	F04	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAGTGTAGCACCCAAGACCACTCTCCGG
FBC	F05	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCGAGCAACACCCAAGACCACTCTCCGG
FBC	F06	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTACTCCCACCCAAGACCACTCTCCGG
FBC	F07	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGATGCCACACCCAAGACCACTCTCCGG
FBC	F08	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGACCGATCACCCAAGACCACTCTCCGG
FBC	F09	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGACGTTGGCACCCAAGACCACTCTCCGG
FBC	F10	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGATGAGCGCACCCAAGACCACTCTCCGG
FBC	F11	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTACTCCGCACCCAAGACCACTCTCCGG
FBC	F12	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGATTCACCACCCAAGACCACTCTCCGG
FBC	G01	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGATGACGCCACCCAAGACCACTCTCCGG
FBC	G02	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGGTTGTTCACCCAAGACCACTCTCCGG
FBC	G03	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGTACTTGCACCCAAGACCACTCTCCGG
FBC	G04	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTAGCAAGCACCCAAGACCACTCTCCGG

FBC	G05	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTGCCATCACCCAAGACCACTCTCCGG
FBC	G06	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGAGAACACACCCAAGACCACTCTCCGG
FBC	G07	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGTATAGCCACCCAAGACCACTCTCCGG
FBC	G08	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTGATGGACACCCAAGACCACTCTCCGG
FBC	G09	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGGCAGTACACCCAAGACCACTCTCCGG
FBC	G10	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGAAGAAGCACCCAAGACCACTCTCCGG
FBC	G11	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAGCGGTTCACCCAAGACCACTCTCCGG
FBC	G12	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTAAGGCCCACCCAAGACCACTCTCCGG
FBC	H01	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAACCTGTCACCCAAGACCACTCTCCGG
FBC	H02	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGAGTACACCACCCAAGACCACTCTCCGG
FBC	H03	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTCGTAGCACCCAAGACCACTCTCCGG
FBC	H04	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTAGGTGCACCCAAGACCACTCTCCGG
FBC	H05	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCGATACCCACCCAAGACCACTCTCCGG
FBC	H06	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGTCGGCTACACCCAAGACCACTCTCCGG
FBC	H07	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCGGTTGTCACCCAAGACCACTCTCCGG
FBC	H08	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGATTGCCTCACCCAAGACCACTCTCCGG
FBC	H09	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCATTCGACACCCAAGACCACTCTCCGG
FBC	H10	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGCACAATCACCCAAGACCACTCTCCGG
FBC	H11	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGGCAGTAACACCCAAGACCACTCTCCGG
FBC	H12	TCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCCTAATCCACCCAAGACCACTCTCCGG
RBC	A01	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGAACTGCCGGTGTGCGAAGTAGGTGC
RBC	A02	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACCAGGTCGGTGTGCGAAGTAGGTGC
RBC	A03	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCTAGAGCGGTGTGCGAAGTAGGTGC
RBC	A04	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCACACAACGGTGTGCGAAGTAGGTGC
RBC	A05	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGTGGAACCGGTGTGCGAAGTAGGTGC
RBC	A06	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGATATGCCCGGTGTGCGAAGTAGGTGC
RBC	A07	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGGTCTGACGGTGTGCGAAGTAGGTGC
RBC	A08	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGTGAGATCGGTGTGCGAAGTAGGTGC
RBC	A09	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTTGGCAGCGGTGTGCGAAGTAGGTGC
RBC	A10	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGATGCCTGCGGTGTGCGAAGTAGGTGC
RBC	A11	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCCGAAGCGGTGTGCGAAGTAGGTGC
RBC	A12	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGGCTTACCGGTGTGCGAAGTAGGTGC
RBC	B01	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGTTGGCCGGTGTGCGAAGTAGGTGC
RBC	B02	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAACGATGCGGTGTGCGAAGTAGGTGC
RBC	B03	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACTACCGCGGTGTGCGAAGTAGGTGC
RBC	B04	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGGTGTCTCGGTGTGCGAAGTAGGTGC
RBC	B05	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCCAGCTTCGGTGTGCGAAGTAGGTGC
RBC	B06	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTTAGACGCGGTGTGCGAAGTAGGTGC
RBC	B07	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACCATACCGGTGTGCGAAGTAGGTGC
RBC	B08	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGACGACTCGGTGTGCGAAGTAGGTGC
RBC	B09	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGTCACCTCGGTGTGCGAAGTAGGTGC

RBC	B10	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCGTGATGCGGTGTGCGAAGTAGGTGC
RBC	B11	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGCTTCCTCGGTGTGCGAAGTAGGTGC
RBC	B12	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTAGACGTCGGTGTGCGAAGTAGGTGC
RBC	C01	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCGGACTTCGGTGTGCGAAGTAGGTGC
RBC	C02	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACCGGAACGGTGTGCGAAGTAGGTGC
RBC	C03	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCCGAAGTCGGTGTGCGAAGTAGGTGC
RBC	C04	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCACGCACGGTGTGCGAAGTAGGTGC
RBC	C05	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGATCCTCGCGGTGTGCGAAGTAGGTGC
RBC	C06	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCGAATAGCGGTGTGCGAAGTAGGTGC
RBC	C07	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTATCCGGCGGTGTGCGAAGTAGGTGC
RBC	C08	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGCAAGACGGTGTGCGAAGTAGGTGC
RBC	C09	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTGTCGACCGGTGTGCGAAGTAGGTGC
RBC	C10	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTTCCATGCGGTGTGCGAAGTAGGTGC
RBC	C11	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGCAATCGCGGTGTGCGAAGTAGGTGC
RBC	C12	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTGAGTGGCGGTGTGCGAAGTAGGTGC
RBC	D01	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTAGGAGACGGTGTGCGAAGTAGGTGC
RBC	D02	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGTCAGTCGGTGTGCGAAGTAGGTGC
RBC	D03	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGTGCTGTCGGTGTGCGAAGTAGGTGC
RBC	D04	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCAACAACCGGTGTGCGAAGTAGGTGC
RBC	D05	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAATAGCCCGGTGTGCGAAGTAGGTGC
RBC	D06	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCTGTGACGGTGTGCGAAGTAGGTGC
RBC	D07	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTGTGGTACGGTGTGCGAAGTAGGTGC
RBC	D08	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGCGTATGCGGTGTGCGAAGTAGGTGC
RBC	D09	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGTTACGCGGTGTGCGAAGTAGGTGC
RBC	D10	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTTCCTGCCGGTGTGCGAAGTAGGTGC
RBC	D11	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTATGTCGCGGTGTGCGAAGTAGGTGC
RBC	D12	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGGAGAGACGGTGTGCGAAGTAGGTGC
RBC	E01	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCCTTAGGCGGTGTGCGAAGTAGGTGC
RBC	E02	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTGTATCCCGGTGTGCGAAGTAGGTGC
RBC	E03	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCAACCTGCGGTGTGCGAAGTAGGTGC
RBC	E04	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCTGATGACGGTGTGCGAAGTAGGTGC
RBC	E05	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAAGACAGCGGTGTGCGAAGTAGGTGC
RBC	E06	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGCTCGTCGGTGTGCGAAGTAGGTGC
RBC	E07	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGATTGCGCGGTGTGCGAAGTAGGTGC
RBC	E08	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCCTTCACGGTGTGCGAAGTAGGTGC
RBC	E09	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCACAGGCGGTGTGCGAAGTAGGTGC
RBC	E10	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGAGCTGCGGTGTGCGAAGTAGGTGC
RBC	E11	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCCTCTGTCGGTGTGCGAAGTAGGTGC
RBC	E12	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCCTCGAACGGTGTGCGAAGTAGGTGC
RBC	F01	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGTGTCTCCGGTGTGCGAAGTAGGTGC
RBC	F02	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGATTGAGGCGGTGTGCGAAGTAGGTGC

RBC	F03	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGACAATCCGGTGTGCGAAGTAGGTGC
RBC	F04	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCACTTGCCGGTGTGCGAAGTAGGTGC
RBC	F05	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTGAACGCCGGTGTGCGAAGTAGGTGC
RBC	F06	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCGTAGCACGGTGTGCGAAGTAGGTGC
RBC	F07	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGGTTCCCGGTGTGCGAAGTAGGTGC
RBC	F08	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGTACACACGGTGTGCGAAGTAGGTGC
RBC	F09	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGATAGGTCGGTGTGCGAAGTAGGTGC
RBC	F10	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTAGCCTCCGGTGTGCGAAGTAGGTGC
RBC	F11	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTTCAGCCCGGTGTGCGAAGTAGGTGC
RBC	F12	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGGGATTCACGGTGTGCGAAGTAGGTGC
RBC	G01	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTGAGCCTCGGTGTGCGAAGTAGGTGC
RBC	G02	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAACGCGACGGTGTGCGAAGTAGGTGC
RBC	G03	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCATTGCCGGTGTGCGAAGTAGGTGC
RBC	G04	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGCATCTCGGTGTGCGAAGTAGGTGC
RBC	G05	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTTGGTCTCGGTGTGCGAAGTAGGTGC
RBC	G06	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCAAGGATCGGTGTGCGAAGTAGGTGC
RBC	G07	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGACGTCCGGTGTGCGAAGTAGGTGC
RBC	G08	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAGGTCAACGGTGTGCGAAGTAGGTGC
RBC	G09	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGATGCTACCGGTGTGCGAAGTAGGTGC
RBC	G10	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCTCTGATCGGTGTGCGAAGTAGGTGC
RBC	G11	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCAAGTCCGGTGTGCGAAGTAGGTGC
RBC	G12	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCGAGCTCGGTGTGCGAAGTAGGTGC
RBC	H01	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACAGTCTCGGTGTGCGAAGTAGGTGC
RBC	H02	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCAGATACCGGTGTGCGAAGTAGGTGC
RBC	H03	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTACGTTCCGGTGTGCGAAGTAGGTGC
RBC	H04	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACGGTTCCGGTGTGCGAAGTAGGTGC
RBC	H05	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCATCGTCCGGTGTGCGAAGTAGGTGC
RBC	H06	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTACGCATCGGTGTGCGAAGTAGGTGC
RBC	H07	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGCTTAGACCGGTGTGCGAAGTAGGTGC
RBC	H08	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGAACTGACCGGTGTGCGAAGTAGGTGC
RBC	H09	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACTTGCACGGTGTGCGAAGTAGGTGC
RBC	H10	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACGCGATCGGTGTGCGAAGTAGGTGC
RBC	H11	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGTCGACACCGGTGTGCGAAGTAGGTGC
RBC	H12	GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAGACTCAACCGGTGTGCGAAGTAGGTGC

B.5 Dual-Indexing Platemaps

This section contains all platemaps for the dual indexing plates (DI plates) used in Chapter 4. The tables that follow show how the primers from the forward and reverse barcode plates (Table B-2) were arrayed to produce the barcode plates. Each entry in the below platemaps follows the format "Well-Barcode Plate," where the "-" delimits the plate and well. An "F" after the delimiter indicates that the well preceding the delimiter was from the forward barcode plate ("FBC" in Table B-2) and an "R" indicates that the well was from the reverse barcode plate ("RBC"). A detailed protocol for how the dual index plates were produced is given in Section B.2.2.

Table B-3. Platemap for DI01 used in Chapter 4.

DI01	1	2	3	4	5	6	7	8	9	10	11	12
A	A01-F, A01-R	A02-F, A02-R	A03-F, A03-R	A04-F, A04-R	A05-F, A05-R	A06-F, A06-R	A07-F, A07-R	A08-F, A08-R	A09-F, A09-R	A10-F, A10-R	A11-F, A11-R	A12-F, A12-R
B	B01-F, B01-R	B02-F, B02-R	B03-F, B03-R	B04-F, B04-R	B05-F, B05-R	B06-F, B06-R	B07-F, B07-R	B08-F, B08-R	B09-F, B09-R	B10-F, B10-R	B11-F, B11-R	B12-F, B12-R
C	C01-F, C01-R	C02-F, C02-R	C03-F, C03-R	C04-F, C04-R	C05-F, C05-R	C06-F, C06-R	C07-F, C07-R	C08-F, C08-R	C09-F, C09-R	C10-F, C10-R	C11-F, C11-R	C12-F, C12-R
D	D01-F, D01-R	D02-F, D02-R	D03-F, D03-R	D04-F, D04-R	D05-F, D05-R	D06-F, D06-R	D07-F, D07-R	D08-F, D08-R	D09-F, D09-R	D10-F, D10-R	D11-F, D11-R	D12-F, D12-R
E	E01-F, E01-R	E02-F, E02-R	E03-F, E03-R	E04-F, E04-R	E05-F, E05-R	E06-F, E06-R	E07-F, E07-R	E08-F, E08-R	E09-F, E09-R	E10-F, E10-R	E11-F, E11-R	E12-F, E12-R
F	F01-F, F01-R	F02-F, F02-R	F03-F, F03-R	F04-F, F04-R	F05-F, F05-R	F06-F, F06-R	F07-F, F07-R	F08-F, F08-R	F09-F, F09-R	F10-F, F10-R	F11-F, F11-R	F12-F, F12-R
G	G01-F, G01-R	G02-F, G02-R	G03-F, G03-R	G04-F, G04-R	G05-F, G05-R	G06-F, G06-R	G07-F, G07-R	G08-F, G08-R	G09-F, G09-R	G10-F, G10-R	G11-F, G11-R	G12-F, G12-R
H	H01-F, H01-R	H02-F, H02-R	H03-F, H03-R	H04-F, H04-R	H05-F, H05-R	H06-F, H06-R	H07-F, H07-R	H08-F, H08-R	H09-F, H09-R	H10-F, H10-R	H11-F, H11-R	H12-F, H12-R

Table B-4. Platemap for DI02 used in Chapter 4.

DI02	1	2	3	4	5	6	7	8	9	10	11	12
A	A01-F, H01-R	A02-F, H02-R	A03-F, H03-R	A04-F, H04-R	A05-F, H05-R	A06-F, H06-R	A07-F, H07-R	A08-F, H08-R	A09-F, H09-R	A10-F, H10-R	A11-F, H11-R	A12-F, H12-R
B	B01-F, A01-R	B02-F, A02-R	B03-F, A03-R	B04-F, A04-R	B05-F, A05-R	B06-F, A06-R	B07-F, A07-R	B08-F, A08-R	B09-F, A09-R	B10-F, A10-R	B11-F, A11-R	B12-F, A12-R
C	C01-F, B01-R	C02-F, B02-R	C03-F, B03-R	C04-F, B04-R	C05-F, B05-R	C06-F, B06-R	C07-F, B07-R	C08-F, B08-R	C09-F, B09-R	C10-F, B10-R	C11-F, B11-R	C12-F, B12-R
D	D01-F, C01-R	D02-F, C02-R	D03-F, C03-R	D04-F, C04-R	D05-F, C05-R	D06-F, C06-R	D07-F, C07-R	D08-F, C08-R	D09-F, C09-R	D10-F, C10-R	D11-F, C11-R	D12-F, C12-R
E	E01-F, D01-R	E02-F, D02-R	E03-F, D03-R	E04-F, D04-R	E05-F, D05-R	E06-F, D06-R	E07-F, D07-R	E08-F, D08-R	E09-F, D09-R	E10-F, D10-R	E11-F, D11-R	E12-F, D12-R
F	F01-F, E01-R	F02-F, E02-R	F03-F, E03-R	F04-F, E04-R	F05-F, E05-R	F06-F, E06-R	F07-F, E07-R	F08-F, E08-R	F09-F, E09-R	F10-F, E10-R	F11-F, E11-R	F12-F, E12-R
G	G01-F, F01-R	G02-F, F02-R	G03-F, F03-R	G04-F, F04-R	G05-F, F05-R	G06-F, F06-R	G07-F, F07-R	G08-F, F08-R	G09-F, F09-R	G10-F, F10-R	G11-F, F11-R	G12-F, F12-R
H	H01-F, G01-R	H02-F, G02-R	H03-F, G03-R	H04-F, G04-R	H05-F, G05-R	H06-F, G06-R	H07-F, G07-R	H08-F, G08-R	H09-F, G09-R	H10-F, G10-R	H11-F, G11-R	H12-F, G12-R

Table B-5. Platemap for DI03 used in Chapter 4.

DI03	1	2	3	4	5	6	7	8	9	10	11	12
A	A01-F, G01-R	A02-F, G02-R	A03-F, G03-R	A04-F, G04-R	A05-F, G05-R	A06-F, G06-R	A07-F, G07-R	A08-F, G08-R	A09-F, G09-R	A10-F, G10-R	A11-F, G11-R	A12-F, G12-R
B	B01-F, H01-R	B02-F, H02-R	B03-F, H03-R	B04-F, H04-R	B05-F, H05-R	B06-F, H06-R	B07-F, H07-R	B08-F, H08-R	B09-F, H09-R	B10-F, H10-R	B11-F, H11-R	B12-F, H12-R
C	C01-F, A01-R	C02-F, A02-R	C03-F, A03-R	C04-F, A04-R	C05-F, A05-R	C06-F, A06-R	C07-F, A07-R	C08-F, A08-R	C09-F, A09-R	C10-F, A10-R	C11-F, A11-R	C12-F, A12-R
D	D01-F, B01-R	D02-F, B02-R	D03-F, B03-R	D04-F, B04-R	D05-F, B05-R	D06-F, B06-R	D07-F, B07-R	D08-F, B08-R	D09-F, B09-R	D10-F, B10-R	D11-F, B11-R	D12-F, B12-R
E	E01-F, C01-R	E02-F, C02-R	E03-F, C03-R	E04-F, C04-R	E05-F, C05-R	E06-F, C06-R	E07-F, C07-R	E08-F, C08-R	E09-F, C09-R	E10-F, C10-R	E11-F, C11-R	E12-F, C12-R
F	F01-F, D01-R	F02-F, D02-R	F03-F, D03-R	F04-F, D04-R	F05-F, D05-R	F06-F, D06-R	F07-F, D07-R	F08-F, D08-R	F09-F, D09-R	F10-F, D10-R	F11-F, D11-R	F12-F, D12-R
G	G01-F, E01-R	G02-F, E02-R	G03-F, E03-R	G04-F, E04-R	G05-F, E05-R	G06-F, E06-R	G07-F, E07-R	G08-F, E08-R	G09-F, E09-R	G10-F, E10-R	G11-F, E11-R	G12-F, E12-R
H	H01-F, F01-R	H02-F, F02-R	H03-F, F03-R	H04-F, F04-R	H05-F, F05-R	H06-F, F06-R	H07-F, F07-R	H08-F, F08-R	H09-F, F09-R	H10-F, F10-R	H11-F, F11-R	H12-F, F12-R

Table B-6. Platemap for DI04 used in Chapter 4.

DI04	1	2	3	4	5	6	7	8	9	10	11	12
A	A01-F, F01-R	A02-F, F02-R	A03-F, F03-R	A04-F, F04-R	A05-F, F05-R	A06-F, F06-R	A07-F, F07-R	A08-F, F08-R	A09-F, F09-R	A10-F, F10-R	A11-F, F11-R	A12-F, F12-R
B	B01-F, G01-R	B02-F, G02-R	B03-F, G03-R	B04-F, G04-R	B05-F, G05-R	B06-F, G06-R	B07-F, G07-R	B08-F, G08-R	B09-F, G09-R	B10-F, G10-R	B11-F, G11-R	B12-F, G12-R
C	C01-F, H01-R	C02-F, H02-R	C03-F, H03-R	C04-F, H04-R	C05-F, H05-R	C06-F, H06-R	C07-F, H07-R	C08-F, H08-R	C09-F, H09-R	C10-F, H10-R	C11-F, H11-R	C12-F, H12-R
D	D01-F, A01-R	D02-F, A02-R	D03-F, A03-R	D04-F, A04-R	D05-F, A05-R	D06-F, A06-R	D07-F, A07-R	D08-F, A08-R	D09-F, A09-R	D10-F, A10-R	D11-F, A11-R	D12-F, A12-R
E	E01-F, B01-R	E02-F, B02-R	E03-F, B03-R	E04-F, B04-R	E05-F, B05-R	E06-F, B06-R	E07-F, B07-R	E08-F, B08-R	E09-F, B09-R	E10-F, B10-R	E11-F, B11-R	E12-F, B12-R
F	F01-F, C01-R	F02-F, C02-R	F03-F, C03-R	F04-F, C04-R	F05-F, C05-R	F06-F, C06-R	F07-F, C07-R	F08-F, C08-R	F09-F, C09-R	F10-F, C10-R	F11-F, C11-R	F12-F, C12-R
G	G01-F, D01-R	G02-F, D02-R	G03-F, D03-R	G04-F, D04-R	G05-F, D05-R	G06-F, D06-R	G07-F, D07-R	G08-F, D08-R	G09-F, D09-R	G10-F, D10-R	G11-F, D11-R	G12-F, D12-R
H	H01-F, E01-R	H02-F, E02-R	H03-F, E03-R	H04-F, E04-R	H05-F, E05-R	H06-F, E06-R	H07-F, E07-R	H08-F, E08-R	H09-F, E09-R	H10-F, E10-R	H11-F, E11-R	H12-F, E12-R

Table B-7. Platemap for DI05 used in Chapter 4.

DI05	1	2	3	4	5	6	7	8	9	10	11	12
A	A01-F, E01-R	A02-F, E02-R	A03-F, E03-R	A04-F, E04-R	A05-F, E05-R	A06-F, E06-R	A07-F, E07-R	A08-F, E08-R	A09-F, E09-R	A10-F, E10-R	A11-F, E11-R	A12-F, E12-R
B	B01-F, F01-R	B02-F, F02-R	B03-F, F03-R	B04-F, F04-R	B05-F, F05-R	B06-F, F06-R	B07-F, F07-R	B08-F, F08-R	B09-F, F09-R	B10-F, F10-R	B11-F, F11-R	B12-F, F12-R
C	C01-F, G01-R	C02-F, G02-R	C03-F, G03-R	C04-F, G04-R	C05-F, G05-R	C06-F, G06-R	C07-F, G07-R	C08-F, G08-R	C09-F, G09-R	C10-F, G10-R	C11-F, G11-R	C12-F, G12-R
D	D01-F, H01-R	D02-F, H02-R	D03-F, H03-R	D04-F, H04-R	D05-F, H05-R	D06-F, H06-R	D07-F, H07-R	D08-F, H08-R	D09-F, H09-R	D10-F, H10-R	D11-F, H11-R	D12-F, H12-R
E	E01-F, A01-R	E02-F, A02-R	E03-F, A03-R	E04-F, A04-R	E05-F, A05-R	E06-F, A06-R	E07-F, A07-R	E08-F, A08-R	E09-F, A09-R	E10-F, A10-R	E11-F, A11-R	E12-F, A12-R
F	F01-F, B01-R	F02-F, B02-R	F03-F, B03-R	F04-F, B04-R	F05-F, B05-R	F06-F, B06-R	F07-F, B07-R	F08-F, B08-R	F09-F, B09-R	F10-F, B10-R	F11-F, B11-R	F12-F, B12-R
G	G01-F, C01-R	G02-F, C02-R	G03-F, C03-R	G04-F, C04-R	G05-F, C05-R	G06-F, C06-R	G07-F, C07-R	G08-F, C08-R	G09-F, C09-R	G10-F, C10-R	G11-F, C11-R	G12-F, C12-R
H	H01-F, D01-R	H02-F, D02-R	H03-F, D03-R	H04-F, D04-R	H05-F, D05-R	H06-F, D06-R	H07-F, D07-R	H08-F, D08-R	H09-F, D09-R	H10-F, D10-R	H11-F, D11-R	H12-F, D12-R

Table B-8. Platemap for DI06 used in Chapter 4.

DI06	1	2	3	4	5	6	7	8	9	10	11	12
A	A01-F, D01-R	A02-F, D02-R	A03-F, D03-R	A04-F, D04-R	A05-F, D05-R	A06-F, D06-R	A07-F, D07-R	A08-F, D08-R	A09-F, D09-R	A10-F, D10-R	A11-F, D11-R	A12-F, D12-R
B	B01-F, E01-R	B02-F, E02-R	B03-F, E03-R	B04-F, E04-R	B05-F, E05-R	B06-F, E06-R	B07-F, E07-R	B08-F, E08-R	B09-F, E09-R	B10-F, E10-R	B11-F, E11-R	B12-F, E12-R
C	C01-F, F01-R	C02-F, F02-R	C03-F, F03-R	C04-F, F04-R	C05-F, F05-R	C06-F, F06-R	C07-F, F07-R	C08-F, F08-R	C09-F, F09-R	C10-F, F10-R	C11-F, F11-R	C12-F, F12-R
D	D01-F, G01-R	D02-F, G02-R	D03-F, G03-R	D04-F, G04-R	D05-F, G05-R	D06-F, G06-R	D07-F, G07-R	D08-F, G08-R	D09-F, G09-R	D10-F, G10-R	D11-F, G11-R	D12-F, G12-R
E	E01-F, H01-R	E02-F, H02-R	E03-F, H03-R	E04-F, H04-R	E05-F, H05-R	E06-F, H06-R	E07-F, H07-R	E08-F, H08-R	E09-F, H09-R	E10-F, H10-R	E11-F, H11-R	E12-F, H12-R
F	F01-F, A01-R	F02-F, A02-R	F03-F, A03-R	F04-F, A04-R	F05-F, A05-R	F06-F, A06-R	F07-F, A07-R	F08-F, A08-R	F09-F, A09-R	F10-F, A10-R	F11-F, A11-R	F12-F, A12-R
G	G01-F, B01-R	G02-F, B02-R	G03-F, B03-R	G04-F, B04-R	G05-F, B05-R	G06-F, B06-R	G07-F, B07-R	G08-F, B08-R	G09-F, B09-R	G10-F, B10-R	G11-F, B11-R	G12-F, B12-R
H	H01-F, C01-R	H02-F, C02-R	H03-F, C03-R	H04-F, C04-R	H05-F, C05-R	H06-F, C06-R	H07-F, C07-R	H08-F, C08-R	H09-F, C09-R	H10-F, C10-R	H11-F, C11-R	H12-F, C12-R

Table B-9. Platemap for DI07 used in Chapter 4.

DI07	1	2	3	4	5	6	7	8	9	10	11	12
A	A01-F, C01-R	A02-F, C02-R	A03-F, C03-R	A04-F, C04-R	A05-F, C05-R	A06-F, C06-R	A07-F, C07-R	A08-F, C08-R	A09-F, C09-R	A10-F, C10-R	A11-F, C11-R	A12-F, C12-R
B	B01-F, D01-R	B02-F, D02-R	B03-F, D03-R	B04-F, D04-R	B05-F, D05-R	B06-F, D06-R	B07-F, D07-R	B08-F, D08-R	B09-F, D09-R	B10-F, D10-R	B11-F, D11-R	B12-F, D12-R
C	C01-F, E01-R	C02-F, E02-R	C03-F, E03-R	C04-F, E04-R	C05-F, E05-R	C06-F, E06-R	C07-F, E07-R	C08-F, E08-R	C09-F, E09-R	C10-F, E10-R	C11-F, E11-R	C12-F, E12-R
D	D01-F, F01-R	D02-F, F02-R	D03-F, F03-R	D04-F, F04-R	D05-F, F05-R	D06-F, F06-R	D07-F, F07-R	D08-F, F08-R	D09-F, F09-R	D10-F, F10-R	D11-F, F11-R	D12-F, F12-R
E	E01-F, G01-R	E02-F, G02-R	E03-F, G03-R	E04-F, G04-R	E05-F, G05-R	E06-F, G06-R	E07-F, G07-R	E08-F, G08-R	E09-F, G09-R	E10-F, G10-R	E11-F, G11-R	E12-F, G12-R
F	F01-F, H01-R	F02-F, H02-R	F03-F, H03-R	F04-F, H04-R	F05-F, H05-R	F06-F, H06-R	F07-F, H07-R	F08-F, H08-R	F09-F, H09-R	F10-F, H10-R	F11-F, H11-R	F12-F, H12-R
G	G01-F, A01-R	G02-F, A02-R	G03-F, A03-R	G04-F, A04-R	G05-F, A05-R	G06-F, A06-R	G07-F, A07-R	G08-F, A08-R	G09-F, A09-R	G10-F, A10-R	G11-F, A11-R	G12-F, A12-R
H	H01-F, B01-R	H02-F, B02-R	H03-F, B03-R	H04-F, B04-R	H05-F, B05-R	H06-F, B06-R	H07-F, B07-R	H08-F, B08-R	H09-F, B09-R	H10-F, B10-R	H11-F, B11-R	H12-F, B12-R

Table B-10. Platemap for DI08 used in Chapter 4.

DI08	1	2	3	4	5	6	7	8	9	10	11	12
A	A01-F, B01-R	A02-F, B02-R	A03-F, B03-R	A04-F, B04-R	A05-F, B05-R	A06-F, B06-R	A07-F, B07-R	A08-F, B08-R	A09-F, B09-R	A10-F, B10-R	A11-F, B11-R	A12-F, B12-R
B	B01-F, C01-R	B02-F, C02-R	B03-F, C03-R	B04-F, C04-R	B05-F, C05-R	B06-F, C06-R	B07-F, C07-R	B08-F, C08-R	B09-F, C09-R	B10-F, C10-R	B11-F, C11-R	B12-F, C12-R
C	C01-F, D01-R	C02-F, D02-R	C03-F, D03-R	C04-F, D04-R	C05-F, D05-R	C06-F, D06-R	C07-F, D07-R	C08-F, D08-R	C09-F, D09-R	C10-F, D10-R	C11-F, D11-R	C12-F, D12-R
D	D01-F, E01-R	D02-F, E02-R	D03-F, E03-R	D04-F, E04-R	D05-F, E05-R	D06-F, E06-R	D07-F, E07-R	D08-F, E08-R	D09-F, E09-R	D10-F, E10-R	D11-F, E11-R	D12-F, E12-R
E	E01-F, F01-R	E02-F, F02-R	E03-F, F03-R	E04-F, F04-R	E05-F, F05-R	E06-F, F06-R	E07-F, F07-R	E08-F, F08-R	E09-F, F09-R	E10-F, F10-R	E11-F, F11-R	E12-F, F12-R
F	F01-F, G01-R	F02-F, G02-R	F03-F, G03-R	F04-F, G04-R	F05-F, G05-R	F06-F, G06-R	F07-F, G07-R	F08-F, G08-R	F09-F, G09-R	F10-F, G10-R	F11-F, G11-R	F12-F, G12-R
G	G01-F, H01-R	G02-F, H02-R	G03-F, H03-R	G04-F, H04-R	G05-F, H05-R	G06-F, H06-R	G07-F, H07-R	G08-F, H08-R	G09-F, H09-R	G10-F, H10-R	G11-F, H11-R	G12-F, H12-R
H	H01-F, A01-R	H02-F, A02-R	H03-F, A03-R	H04-F, A04-R	H05-F, A05-R	H06-F, A06-R	H07-F, A07-R	H08-F, A08-R	H09-F, A09-R	H10-F, A10-R	H11-F, A11-R	H12-F, A12-R

B.6 Supplemental Tables

Table B-11. evSeq captures off-target mutations. This table is derived from the "AminoAcids_Coupled_Max.csv" output file from evSeq for the TrpB run presented in Chapter 4, and shows all confident (defined as ≥0.80 alignment frequency and ≥10 total reads) unexpected mutations captured by evSeq; some columns have been removed. Note in the "VariantCombo" column that the amino acid at the expected mutagenized position has a "?" as the original amino acid—this is because the evSeq run generating this data was told the variable positions with the "NNN" convention. For unexpected variable positions, both the original amino acid and the new amino acid are shown.

IndexPlate	Plate	Well	VariantCombo	AlignmentFrequency	WellSeqDepth
DI02	Lib2_118X	E03	?118V_D164G	0.964286	28
DI04	Lib4_166X	B02	P154S_?166Q	0.977011	87
DI08	Lib8_301X	H11	G250D_?301L	0.99537	216

Table B-12. Primer sequences for TrpB saturation mutagenesis library construction in Chapter 4.

Site	Direction	Sequence
105	Forward	GGCAAAACCCGTATCATTGCTNNNACGGGTGCTGGTCAGCAC
105	Reverse	AGCAATGATACGGGTTTTGCCCATTAGTTTTGCCAGCAGAACCTGGC
118	Forward	GGCGTAGCAACTGCTACCNNNGCAGCGCTGTTCGGTATGGAATGTGTAATCTATATGG
118	Reverse	GGTAGCAGTTGCTACGCCGTGCTGACCAGC
162	Forward	GTAAAATCCGGTAGCCGTACCNNNAAAGACGCAATTGACGAAGCTCTG
162	Reverse	GGTACGGCTACCGGATTTTACCGGTACAACTTTAGCACCCAGCAG
166	Forward	CGTACCCTGAAAGACGCANNNGACGAAGCTCTGCGTGACTGGATTACCAACC
166	Reverse	TGCGTCTTTCAGGGTACGGCTACCGGATTTTACCGG
184	Forward	CTGCAGACCACCTATTACGTGNNNGGCTCTGTGGTTGGTCC
184	Reverse	CACGTAATAGGTGGTCTGCAGGTTGGTAATCCAGTCACGCAGAGCT
228	Forward	TACATCGTTGCGTGCGTGNNNGGTGGTTCTAACGCTGCC
228	Reverse	CACGCACGCAACGATGTAGTCCGGCAGACGGCCTTCT
292	Forward	GATGACTGGGGTCAAGTTCAGGTGNNNCACTCCGTCTCCGCTG
292	Reverse	CACCTGAACTTGACCCCAGTCATCCTGCAGAACGAACGTCTTAGAACCG
301	Forward	TCCGCTGGCCTGGACNNNTCCGGTGTCGGTCCGGA
301	Reverse	GTCCAGGCCAGCGGAGACGGAGTGGCTCACCTGAACT

Table B-13. Primers specific to the ampicillin resistance gene of pET22b(+) used in TrpB library construction in Chapter 4.

Site	Direction	Sequence
AmpR	Forward	CCAACTTACTTCTGACAACGATCGGAGGACCGAAGGAGCTAACCGCTTTTTTGC
AmpR	Reverse	CGATCGTTGTCAGAAGTAAGTTGGCCGCAGTGTTATCACTCATGGTTATGGCAG

Table B-14. Inner primers used for evSeq library preparation from the TrpB site-saturation mutagenesis libraries in Chapter 4.

Name	Direction	Sites	Sequence
evSeq_102_f	Forward	105, 118, 162, 166, 184	CACCCAAGACCACTCTCCGGGCAAAA CTAATGGGCAAAACCCG
evSeq_184_r	Reverse	105, 118, 162, 166, 184	CGGTGTGCGAAGTAGGTGCGATGCGG ACCAACCACAGAG
evSeq_226_f	Forward	228, 292, 301	CACCCAAGACCACTCTCCGGGCCGGA CTACATCGTTGCG
evSeq_304_r	Reverse	228, 292, 301	CGGTGTGCGAAGTAGGTGCCAATAGG CGTGTTCCGGACC

Table B-15. The evSeq barcode plates used for sequencing each position of the TrpB site-saturation mutagenesis libraries in Chapter 4.

Position targeted	Barcode plate
105	DI01
118	DI02
162	DI03
166	DI04
184	DI05
228	DI06
292	DI07
301	DI08

Table B-16. Mutagenic primers used for the construction of the *Rma*NOD four-site-saturation library in Chapter 4. Note that the names of the primers are delimited by "-" and that the delimited sections reflect the mutagenized positions, the degenerate codons at those positions, and the direction of the primer on the template DNA ([Positions]-[Codon1]-[Codon2]-[Direction]).

Name	**Sequence**
S28M31-NDT-NDT-F	AAACACTCAGTCGCTATTNDTGCCACGNDTGGTCGGCTGCTTTTCG
S28M31-NDT-VHG-F	AAACACTCAGTCGCTATTNDTGCCACGVHGGGTCGGCTGCTTTTCG
S28M31-NDT-TGG-F	AAACACTCAGTCGCTATTNDTGCCACGTGGGGTCGGCTGCTTTTCG
S28M31-VHG-NDT-F	AAACACTCAGTCGCTATTVHGGCCACGNDTGGTCGGCTGCTTTTCG
S28M31-VHG-VHG-F	AAACACTCAGTCGCTATTVHGGCCACGVHGGGTCGGCTGCTTTTCG
S28M31-VHG-TGG-F	AAACACTCAGTCGCTATTVHGGCCACGTGGGGTCGGCTGCTTTTCG
S28M31-TGG-NDT-F	AAACACTCAGTCGCTATTTGGGCCACGNDTGGTCGGCTGCTTTTCG
S28M31-TGG-VHG-F	AAACACTCAGTCGCTATTTGGGCCACGVHGGGTCGGCTGCTTTTCG
S28M31-TGG-TGG-F	AAACACTCAGTCGCTATTTGGGCCACGTGGGGTCGGCTGCTTTTCG
Q52L56-AHN-AHN-R	GGCCAACAGGGCCGACGCAHNCTTGTGTATAHNTCTCTCAGGAAGTTCAAACAAG
Q52L56-AHN-CDB-R	GGCCAACAGGGCCGACGCAHNCTTGTGTATCDBTCTCTCAGGAAGTTCAAACAAG
Q52L56-AHN-CCA-R	GGCCAACAGGGCCGACGCAHNCTTGTGTATCCATCTCTCAGGAAGTTCAAACAAG
Q52L56-CDB-AHN-R	GGCCAACAGGGCCGACGCCDBCTTGTGTATAHNTCTCTCAGGAAGTTCAAACAAG
Q52L56-CDB-CDB-R	GGCCAACAGGGCCGACGCCDBCTTGTGTATCDBTCTCTCAGGAAGTTCAAACAAG
Q52L56-CDB-CCA-R	GGCCAACAGGGCCGACGCCDBCTTGTGTATCCATCTCTCAGGAAGTTCAAACAAG
Q52L56-CCA-AHN-R	GGCCAACAGGGCCGACGCCCACTTGTGTATAHNTCTCTCAGGAAGTTCAAACAAG
Q52L56-CCA-CDB-R	GGCCAACAGGGCCGACGCCCACTTGTGTATCDBTCTCTCAGGAAGTTCAAACAAG
Q52L56-CCA-CCA-R	GGCCAACAGGGCCGACGCCCACTTGTGTATCCATCTCTCAGGAAGTTCAAACAAG

Table B-17. Additional primers used to build flanking fragments during construction of the four-site-saturation *Rma*NOD library in Chapter 4.

Flanking Fragment	Primer Type	Primer Name	Sequence
0	Forward	Universal-F	CCAACTTACTTCTGACAACGATCGGAG GACCGAAGGAGCTAACCGCTTTTTTGC
0	Reverse	S28M31_Const-R	AATAGCGACTGAGTGTTTCTGCAGTGC AGGCAC
1	Forward	L56_Const-F	GCGTCGGCCCTGTTGGCCTACGCCCGT AGTATCGACAACCC
1	Reverse	Universal-R	CGATCGTTGTCAGAAGTAAGTTGGCCG CAGTGTTATCACTCATGGTTATGGCAG

Table B-18. Inner primers used for evSeq library preparation from the *Rma*NOD four-site-saturation mutagenesis library in Chapter 4.

Plates	Forward primer	Reverse Primer
All plates	CACCCAAGACCACTCTCCGGC ACTGCAGAAACACTCAGTCG	CGGTGTGCGAAGTAGGTGCAC TACGGGCGTAGGCCAAC

Table B-19. The evSeq barcode plates used for sequencing each position of the *Rma*NOD four-site-saturation mutagenesis library in Chapter 4.

Position targeted	Barcode plate
Plate #1	DI01
Plate #2	DI02
Plate #3	DI03
Plate #4	DI04
Plate #5	DI05

REFERENCES

(1) BCC Research Staff. *Global Markets for Enzymes in Industrial Applications*; BCC Research, 2018.

(2) Bornscheuer, U. T.; Huisman, G. W.; Kazlauskas, R. J.; Lutz, S.; Moore, J. C.; Robins, K. Engineering the Third Wave of Biocatalysis. *Nature* **2012**, *485* (7397), 185–194. https://doi.org/10.1038/nature11117.

(3) Blamey, J. M.; Fischer, F.; Meyer, H.-P.; Sarmiento, F.; Zinn, M. Enzymatic Biocatalysis in Chemical Transformations. In *Biotechnology of Microbial Enzymes*; Elsevier, 2017; pp 347–403. https://doi.org/10.1016/B978-0-12-803725-6.00014-5.

(4) Sheldon, R. A.; Woodley, J. M. Role of Biocatalysis in Sustainable Chemistry. *Chem. Rev.* **2018**, *118* (2), 801–838. https://doi.org/10.1021/acs.chemrev.7b00203.

(5) Baker, D. An Exciting but Challenging Road Ahead for Computational Enzyme Design. *Protein Sci.* **2010**, *19* (10), 1817–1819. https://doi.org/10.1002/pro.481.

(6) Leaver-Fay, A.; Tyka, M.; Lewis, S. M.; Lange, O. F.; Thompson, J.; Jacak, R.; Kaufman, K. W.; Renfrew, P. D.; Smith, C. A.; Sheffler, W.; et al. Rosetta3: An Object-Oriented Software Suite for the Simulation and Design of Macromolecules. In *Methods in Enzymology*; 2011; Vol. 487, pp 545–574. https://doi.org/10.1016/B978-0-12-381270-4.00019-6.

(7) Huang, P.-S.; Boyken, S. E.; Baker, D. The Coming of Age of de Novo Protein Design. *Nature* **2016**, *537* (7620), 320–327. https://doi.org/10.1038/nature19946.

(8) Garcia-Borràs, M.; Houk, K. N.; Jiménez-Osés, G. Chapter 4. Computational Design of Protein Function. In *Computational Tools for Chemical Biology*; The Royal Society of Chemistry, 2018; pp 87–107. https://doi.org/10.1039/9781788010139-00087.

(9) Fasan, R.; Jennifer Kan, S. B.; Zhao, H. A Continuing Career in Biocatalysis: Frances H. Arnold. *ACS Catal.* **2019**, *9* (11), 9775–9788. https://doi.org/10.1021/acscatal.9b02737.

(10) Arnold, F. H. Directed Evolution: Bringing New Chemistry to Life. *Angew. Chemie Int. Ed.* **2018**, *57* (16), 4143–4148. https://doi.org/10.1002/anie.201708408.

(11) Wang, Y.; Xue, P.; Cao, M.; Yu, T.; Lane, S. T.; Zhao, H. Directed Evolution: Methodologies and Applications. *Chem. Rev.* **2021**, *121* (20), 12384–12444. https://doi.org/10.1021/acs.chemrev.1c00260.

(12) Anfinsen, C. B.; Haber, E.; Sela, M.; White, F. H. The Kinetics of Formation of Native Ribonuclease During Oxidation of the Reduced Polypeptide Chain. *Proc. Natl. Acad. Sci.* **1961**, *47* (9), 1309–1314. https://doi.org/10.1073/pnas.47.9.1309.

(13) Khersonsky, O.; Tawfik, D. S. Enzyme Promiscuity: A Mechanistic and Evolutionary Perspective. *Annu. Rev. Biochem.* **2010**, *79*, 471–505. https://doi.org/10.1146/annurev-biochem-030409-143718.

(14) Trudeau, D. L.; Tawfik, D. S. Protein Engineers Turned Evolutionists—the Quest for the Optimal Starting Point. *Curr. Opin. Biotechnol.* **2019**, *60*, 46–52. https://doi.org/10.1016/j.copbio.2018.12.002.

(15) Mazurenko, S.; Prokop, Z.; Damborsky, J. Machine Learning in Enzyme Engineering. *ACS Catal.* **2020**, *10* (2), 1210–1223. https://doi.org/10.1021/acscatal.9b04321.

(16) Yang, K. K.; Wu, Z.; Arnold, F. H. Machine-Learning-Guided Directed Evolution for Protein Engineering. *Nat. Methods* **2019**, *16* (8), 687–694. https://doi.org/10.1038/s41592-019-0496-6.

(17) Li, G.; Dong, Y.; Reetz, M. T. Can Machine Learning Revolutionize Directed Evolution of Selective Enzymes? *Adv. Synth. Catal.* **2019**, *361* (11), 2377–2386. https://doi.org/10.1002/adsc.201900149.

(18) Siedhoff, N. E.; Schwaneberg, U.; Davari, M. D. Machine Learning-Assisted Enzyme Engineering. In *Methods in Enzymology*; Elsevier Inc., 2020; Vol. 643, pp 281–315. https://doi.org/10.1016/bs.mie.2020.05.005.

(19) Xu, Y.; Verma, D.; Sheridan, R. P.; Liaw, A.; Ma, J.; Marshall, N. M.; McIntosh, J.; Sherer, E. C.; Svetnik, V.; Johnston, J. M. Deep Dive into Machine Learning Models for Protein Engineering. *J. Chem. Inf. Model.* **2020**, *60* (6), 2773–2790. https://doi.org/10.1021/acs.jcim.0c00073.

(20) AlQuraishi, M.; Sorger, P. K. Differentiable Biology: Using Deep Learning for Biophysics-Based and Data-Driven Modeling of Molecular Mechanisms. *Nat. Methods* **2021**, *18* (10), 1169–1180. https://doi.org/10.1038/s41592-021-01283-4.

(21) Greener, J. G.; Kandathil, S. M.; Moffat, L.; Jones, D. T. A Guide to Machine Learning for Biologists. *Nat. Rev. Mol. Cell Biol.* **2022**, *23* (1), 40–55. https://doi.org/10.1038/s41580-021-00407-0.

(22) Wittmann, B. J.; Johnston, K. E.; Wu, Z.; Arnold, F. H. Advances in Machine Learning for Directed Evolution. *Curr. Opin. Struct. Biol.* **2021**, *69*, 11–18. https://doi.org/10.1016/j.sbi.2021.01.008.

(23) Hie, B. L.; Yang, K. K. Adaptive Machine Learning for Protein Engineering. *Curr. Opin. Struct. Biol.* **2022**, *72*, 145–152. https://doi.org/10.1016/j.sbi.2021.11.002.

(24) Wu, Z.; Johnston, K. E.; Arnold, F. H.; Yang, K. K. Protein Sequence Design with Deep Generative Models. *Curr. Opin. Chem. Biol.* **2021**, *65*, 18–27. https://doi.org/10.1016/j.cbpa.2021.04.004.

(25) Domingos, P. A Few Useful Things to Know about Machine Learning. *Commun. ACM* **2012**, *55* (10), 78–87. https://doi.org/10.1145/2347736.2347755.

(26) Shalev-Shwartz, S.; Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*; Cambridge University Press: Cambridge, 2014; Vol. 128. https://doi.org/10.1017/CBO9781107298019.

(27) Maynard Smith, J. Natural Selection and the Concept of a Protein Space. *Nature* **1970**, *225* (5232), 563–564. https://doi.org/10.1038/225563a0.

(28) Arnold, F. H. The Library of Maynard-Smith: My Search for Meaning in the Protein Universe. *Microbe Mag.* **2011**, *6* (7), 316–318. https://doi.org/10.1128/microbe.6.316.1.

(29) Romero, P. A.; Arnold, F. H. Exploring Protein Fitness Landscapes by Directed Evolution. *Nat. Rev. Mol. Cell Biol.* **2009**, *10* (12), 866–876. https://doi.org/10.1038/nrm2805.

(30) Packer, M. S.; Liu, D. R. Methods for the Directed Evolution of Proteins. *Nat. Rev. Genet.* **2015**, *16* (7), 379–394. https://doi.org/10.1038/nrg3927.

(31) Brookes, D. H.; Aghazadeh, A.; Listgarten, J. On the Sparsity of Fitness Functions and Implications for Learning. *Proc. Natl. Acad. Sci.* **2022**, *119* (1). https://doi.org/10.1073/pnas.2109649118.

(32) Bloom, J. D.; Labthavikul, S. T.; Otey, C. R.; Arnold, F. H. Protein Stability Promotes Evolvability. *Proc. Natl. Acad. Sci.* **2006**, *103* (15), 5869–5874. https://doi.org/10.1073/pnas.0510098103.

(33) Wittmann, B. J.; Yue, Y.; Arnold, F. H. Informed Training Set Design Enables Efficient Machine Learning-Assisted Directed Protein Evolution. *Cell Syst.* **2021**, *12* (11), 1026-1045.e7. https://doi.org/10.1016/j.cels.2021.07.008.

(34) Raschka, S. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. *arXiv* **2018**. https://doi.org/10.48550/arXiv.1811.12808.

(35) Kawashima, S.; Pokarowski, P.; Pokarowska, M.; Kolinski, A.; Katayama, T.; Kanehisa, M. AAindex: Amino Acid Index Database, Progress Report 2008. *Nucleic Acids Res.* **2007**, *36* (Database), D202–D205. https://doi.org/10.1093/nar/gkm998.

(36) Ofer, D.; Linial, M. ProFET: Feature Engineering Captures High-Level Protein Functions. *Bioinformatics* **2015**, *31* (21), 3429–3436. https://doi.org/10.1093/bioinformatics/btv345.

(37) Iuchi, H.; Matsutani, T.; Yamada, K.; Iwano, N.; Sumi, S.; Hosoda, S.; Zhao, S.; Fukunaga, T.; Hamada, M. Representation Learning Applications in Biological Sequence Analysis. *Comput. Struct. Biotechnol. J.* **2021**, *19*, 3198–3208. https://doi.org/10.1016/j.csbj.2021.05.039.

(38) Ibtehaz, N.; Kihara, D. Application of Sequence Embedding in Protein Sequence-Based Predictions. *arXiv* **2021**. https://doi.org/10.48550/arXiv.2110.07609.

(39) Mistry, J.; Chuguransky, S.; Williams, L.; Qureshi, M.; Salazar, G. A.; Sonnhammer, E. L. L.; Tosatto, S. C. E.; Paladin, L.; Raj, S.; Richardson, L. J.; et al. Pfam: The Protein Families Database in 2021. *Nucleic Acids Res.* **2021**, *49* (D1), D412–D419. https://doi.org/10.1093/nar/gkaa913.

(40) Mitchell, A. L.; Almeida, A.; Beracochea, M.; Boland, M.; Burgin, J.; Cochrane, G.; Crusoe, M. R.; Kale, V.; Potter, S. C.; Richardson, L. J.; et al. MGnify: The Microbiome Analysis Resource in 2020. *Nucleic Acids Res.* **2020**, *48* (D1), D570–D578. https://doi.org/10.1093/nar/gkz1035.

(41) Mirdita, M.; von den Driesch, L.; Galiez, C.; Martin, M. J.; Söding, J.; Steinegger, M. Uniclust Databases of Clustered and Deeply Annotated Protein Sequences and Alignments. *Nucleic Acids Res.* **2017**, *45* (D1), D170–D176. https://doi.org/10.1093/nar/gkw1081.

(42) Bateman, A.; Martin, M. J.; Orchard, S.; Magrane, M.; Agivetova, R.; Ahmad, S.; Alpi, E.; Bowler-Barnett, E. H.; Britto, R.; Bursteinas, B.; et al. UniProt: The Universal Protein Knowledgebase in 2021. *Nucleic Acids Res.* **2021**, *49* (D1), D480–D489. https://doi.org/10.1093/nar/gkaa1100.

(43) Liu, Z.; Lin, Y.; Sun, M. *Representation Learning for Natural Language Processing*; Springer Singapore: Singapore, 2020. https://doi.org/10.1007/978-981-15-5573-2.

(44) Galanis, N.-I.; Vafiadis, P.; Mirzaev, K.-G.; Papakostas, G. A. Machine Learning Meets Natural Language Processing -- The Story so Far. *arXiv* **2021**. https://doi.org/10.48550/arXiv.2104.10213.

(45) Young, T.; Hazarika, D.; Poria, S.; Cambria, E. Recent Trends in Deep Learning Based Natural Language Processing. *IEEE Comput. Intell. Mag.* **2018**, *13* (3), 55–75. https://doi.org/10.1109/MCI.2018.2840738.

(46) Ericsson, L.; Gouk, H.; Loy, C. C.; Hospedales, T. M. Self-Supervised Representation Learning: Introduction, Advances and Challenges. *arXiv* **2021**. https://doi.org/10.48550/arXiv.2110.09327.

(47) Di Liello, L.; Gabburo, M.; Moschitti, A. Efficient Pre-Training Objectives for Transformers. *arXiv* **2021**. https://doi.org/10.48550/arXiv.2104.09694.

(48) Vig, J.; Madani, A.; Varshney, L. R.; Xiong, C.; Socher, R.; Rajani, N. F. BERTology Meets Biology: Interpreting Attention in Protein Language Models. *arXiv* **2020**. https://doi.org/10.48550/arXiv.2006.15222.

(49) Melidis, D. P.; Nejdl, W. Capturing Protein Domain Structure and Function Using Self-Supervision on Domain Architectures. *Algorithms* **2021**, *14* (1), 28. https://doi.org/10.3390/a14010028.

(50) Ding, X.; Zou, Z.; Brooks III, C. L. Deciphering Protein Evolution and Fitness Landscapes with Latent Space Models. *Nat. Commun.* **2019**, *10* (1), 5644. https://doi.org/10.1038/s41467-019-13633-0.

(51) Elnaggar, A.; Heinzinger, M.; Dallago, C.; Rehawi, G.; Wang, Y.; Jones, L.; Gibbs, T.; Feher, T.; Angerer, C.; Steinegger, M.; et al. ProtTrans: Towards Cracking the Language of Lifes Code Through Self-Supervised Deep Learning and High Performance Computing. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**. https://doi.org/10.1109/TPAMI.2021.3095381.

(52) Alley, E. C.; Khimulya, G.; Biswas, S.; AlQuraishi, M.; Church, G. M. Unified Rational Protein Engineering with Sequence-Based Deep Representation Learning. *Nat. Methods* **2019**, *16* (12), 1315–1322. https://doi.org/10.1038/s41592-019-0598-1.

(53) Yang, K. K.; Wu, Z.; Bedbrook, C. N.; Arnold, F. H. Learned Protein Embeddings for Machine Learning. *Bioinformatics* **2018**, *34* (15), 2642–2648. https://doi.org/10.1093/bioinformatics/bty178.

(54) Asgari, E.; Mofrad, M. R. K. Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics. *PLoS One* **2015**, *10* (11), e0141287. https://doi.org/10.1371/journal.pone.0141287.

(55) Rives, A.; Meier, J.; Sercu, T.; Goyal, S.; Lin, Z.; Liu, J.; Guo, D.; Ott, M.; Zitnick, C. L.; Ma, J.; et al. Biological Structure and Function Emerge from Scaling Unsupervised Learning to 250 Million Protein Sequences. *Proc. Natl. Acad. Sci.* **2021**, *118* (15). https://doi.org/10.1073/pnas.2016239118.

(56) Heinzinger, M.; Elnaggar, A.; Wang, Y.; Dallago, C.; Nechaev, D.; Matthes, F.; Rost, B. Modeling Aspects of the Language of Life through Transfer-Learning Protein Sequences. *BMC Bioinformatics* **2019**, *20* (1), 723. https://doi.org/10.1186/s12859-019-3220-8.

(57) Madani, A.; McCann, B.; Naik, N.; Keskar, N. S.; Anand, N.; Eguchi, R. R.; Huang, P.-S.; Socher, R. ProGen: Language Modeling for Protein Generation. *arXiv* **2020**. https://doi.org/10.48550/arXiv.2004.03497.

(58) Rao, R.; Bhattacharya, N.; Thomas, N.; Duan, Y.; Chen, X.; Canny, J.; Abbeel, P.; Song, Y. S. Evaluating Protein Transfer Learning with TAPE. *arXiv* **2019**. https://doi.org/10.48550/arXiv.1906.08230.

(59) Kimothi, D.; Soni, A.; Biyani, P.; Hogan, J. M. Distributed Representations for Biological Sequence Analysis. *arXiv* **2016**. https://doi.org/10.48550/arXiv.1608.05949.

(60) Bepler, T.; Berger, B. Learning Protein Sequence Embeddings Using Information from Structure. *arXiv* **2019**. https://doi.org/10.48550/arXiv.1902.08661.

(61) Rao, R.; Liu, J.; Verkuil, R.; Meier, J.; Canny, J. F.; Abbeel, P.; Sercu, T.; Rives, A. MSA Transformer. *bioRxiv* **2021**. https://doi.org/10.1101/2021.02.12.430858.

(62) Shanehsazzadeh, A.; Belanger, D.; Dohan, D. Is Transfer Learning Necessary for Protein Landscape Prediction? *arXiv* **2020**. https://doi.org/10.48550/arXiv.2011.03443.

(63) Hsu, C.; Nisonoff, H.; Fannjiang, C.; Listgarten, J. Learning Protein Fitness Models from Evolutionary and Assay-Labeled Data. *Nat. Biotechnol.* **2022**. https://doi.org/10.1038/s41587-021-01146-5.

(64) Biswas, S.; Khimulya, G.; Alley, E. C.; Esvelt, K. M.; Church, G. M. Low-N Protein Engineering with Data-Efficient Deep Learning. *Nat. Methods* **2021**, *18* (4), 389–396. https://doi.org/10.1038/s41592-021-01100-y.

(65) Lu, T.; Lu, A. X.; Moses, A. M. Random Embeddings and Linear Regression Can Predict Protein Function. *arXiv* **2021**. https://doi.org/10.48550/arXiv.2104.14661.

(66) Detlefsen, N. S.; Hauberg, S.; Boomsma, W. Learning Meaningful Representations of Protein Sequences. *Nat. Commun.* **2022**, *13* (1), 1914. https://doi.org/10.1038/s41467-022-29443-w.

(67) Meier, J.; Rao, R.; Verkuil, R.; Liu, J.; Sercu, T.; Rives, A. Language Models Enable Zero-Shot Prediction of the Effects of Mutations on Protein Function. *bioRxiv* **2021**. https://doi.org/10.1101/2021.07.09.450648.

(68) Riesselman, A. J.; Ingraham, J. B.; Marks, D. S. Deep Generative Models of Genetic Variation Capture the Effects of Mutations. *Nat. Methods* **2018**, *15* (10), 816–822. https://doi.org/10.1038/s41592-018-0138-4.

(69) Khersonsky, O.; Lipsh, R.; Avizemer, Z.; Ashani, Y.; Goldsmith, M.; Leader, H.; Dym, O.; Rogotner, S.; Trudeau, D. L.; Prilusky, J.; et al. Automated Design of Efficient and Functionally Diverse Enzyme Repertoires. *Mol. Cell* **2018**, *72* (1), 178-186.e5. https://doi.org/10.1016/j.molcel.2018.08.033.

(70) Hopf, T. A.; Ingraham, J. B.; Poelwijk, F. J.; Schärfe, C. P. I.; Springer, M.; Sander, C.; Marks, D. S. Mutation Effects Predicted from Sequence Co-Variation. *Nat. Biotechnol.* **2017**, *35* (2), 128–135. https://doi.org/10.1038/nbt.3769.

(71) Sinai, S.; Kelsic, E. A Primer on Model-Guided Exploration of Fitness Landscapes for Biological Sequence Design. *arXiv* **2020**. https://doi.org/10.48550/arXiv.2010.10614.

(72) Sinai, S.; Wang, R.; Whatley, A.; Slocum, S.; Locane, E.; Kelsic, E. D. AdaLead: A Simple and Robust Adaptive Greedy Search Algorithm for Sequence Design. *arXiv* **2020**. https://doi.org/10.48550/arXiv.2010.02141.

(73) Bryant, D. H.; Bashir, A.; Sinai, S.; Jain, N. K.; Ogden, P. J.; Riley, P. F.; Church, G. M.; Colwell, L. J.; Kelsic, E. D. Deep Diversification of an AAV Capsid Protein by Machine Learning. *Nat. Biotechnol.* **2021**, *39* (6), 691–696. https://doi.org/10.1038/s41587-020-00793-4.

(74) Brookes, D. H.; Listgarten, J. Design by Adaptive Sampling. *arXiv* **2018**. https://doi.org/10.48550/arXiv.1810.03714.

(75) Brookes, D. H.; Park, H.; Listgarten, J. Conditioning by Adaptive Sampling for Robust Design. *arXiv* **2020**. https://doi.org/10.48550/arXiv.1901.10060.

(76) Sinai, S.; Jain, N.; Church, G. M.; Kelsic, E. D. Generative AAV Capsid Diversification by Latent Interpolation. *bioRxiv* **2021**. https://doi.org/10.1101/2021.04.16.440236.

(77) Castro, E.; Godavarthi, A.; Rubinfien, J.; Givechian, K. B.; Bhaskar, D.; Krishnaswamy, S. Guided Generative Protein Design Using Regularized Transformers. *arXiv* **2022**. https://doi.org/10.48550/arXiv.2201.09948.

(78) Romero, P. A.; Krause, A.; Arnold, F. H. Navigating the Protein Fitness Landscape with Gaussian Processes. *Proc. Natl. Acad. Sci.* **2013**, *110* (3), E193–E201. https://doi.org/10.1073/pnas.1215251110.

(79) Hie, B.; Bryson, B. D.; Berger, B. Leveraging Uncertainty in Machine Learning Accelerates Biological Discovery and Design. *Cell Syst.* **2020**, *11* (5), 461-477.e9. https://doi.org/10.1016/j.cels.2020.09.007.

(80) 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 Syst.* **2020**, *11* (1), 49-62.e16. https://doi.org/10.1016/j.cels.2020.05.007.

(81) Sanger, F.; Coulson, A. R. A Rapid Method for Determining Sequences in DNA by Primed Synthesis with DNA Polymerase. *J. Mol. Biol.* **1975**, *94* (3), 441–448. https://doi.org/10.1016/0022-2836(75)90213-2.

(82) 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.* **2019**, *116* (18), 8852–8858. https://doi.org/10.1073/pnas.1901979116.

(83) Metzker, M. L. Sequencing Technologies — the next Generation. *Nat. Rev. Genet.* **2010**, *11* (1), 31–46. https://doi.org/10.1038/nrg2626.

(84) Wittmann, B. J.; Johnston, K. E.; Almhjell, P. J.; Arnold, F. H. EvSeq: Cost-Effective Amplicon Sequencing of Every Variant in a Protein Library. *ACS Synth. Biol.* **2022**, *11* (3), 1313–1324. https://doi.org/10.1021/acssynbio.1c00592.

(85) Kille, S.; Acevedo-Rocha, C. G.; Parra, L. P.; Zhang, Z.-G.; Opperman, D. J.; Reetz, M. T.; Acevedo, J. P. Reducing Codon Redundancy and Screening Effort of Combinatorial Protein Libraries Created by Saturation Mutagenesis. *ACS Synth. Biol.* **2013**, *2* (2), 83–92. https://doi.org/10.1021/sb300037w.

(86) Jacobs, T. M.; Yumerefendi, H.; Kuhlman, B.; Leaver-Fay, A. SwiftLib: Rapid Degenerate-Codon-Library Optimization through Dynamic Programming. *Nucleic Acids Res.* **2015**, *43* (5), e34. https://doi.org/10.1093/nar/gku1323.

(87) Shimko, T. C.; Fordyce, P. M.; Orenstein, Y. DeCoDe: Degenerate Codon Design for Complete Protein-Coding DNA Libraries. *Bioinformatics* **2020**, *36* (11), 3357–3364. https://doi.org/10.1093/bioinformatics/btaa162.

(88) Tretyachenko, V.; Voráček, V.; Souček, R.; Fujishima, K.; Hlouchová, K. CoLiDe: Combinatorial Library Design Tool for Probing Protein Sequence Space. *Bioinformatics* **2021**, *37* (4), 482–489. https://doi.org/10.1093/bioinformatics/btaa804.

(89) David, B. M.; Wyllie, R. M.; Harouaka, R.; Jensen, P. A. A Reinforcement Learning Framework for Pooled Oligonucleotide Design. *Bioinformatics* **2022**, *38* (8), 2219–2225. https://doi.org/10.1093/bioinformatics/btac073.

(90) Weinstein, E. N.; Amin, A. N.; Grathwohl, W.; Kassler, D.; Disset, J.; Marks, D. S. Optimal Design of Stochastic DNA Synthesis Protocols Based on Generative Sequence Models. *bioRxiv* **2021**. https://doi.org/10.1101/2021.10.28.466307.

(91) Cao, L.; Goreshnik, I.; Coventry, B.; Case, J. B.; Miller, L.; Kozodoy, L.; Chen, R. E.; Carter, L.; Walls, A. C.; Park, Y.-J.; et al. De Novo Design of Picomolar SARS-CoV-2 Miniprotein Inhibitors. *Science* **2020**, *370* (6515), 426–431. https://doi.org/10.1126/science.abd9909.

(92) Villegas-Morcillo, A.; Makrodimitris, S.; van Ham, R. C. H. J.; Gomez, A. M.; Sanchez, V.; Reinders, M. J. T. Unsupervised Protein Embeddings Outperform Hand-Crafted Sequence and Structure Features at Predicting Molecular Function. *Bioinformatics* **2021**, *37* (2), 162–170. https://doi.org/10.1093/bioinformatics/btaa701.

(93) Heinzinger, M.; Littmann, M.; Sillitoe, I.; Bordin, N.; Orengo, C.; Rost, B. Contrastive Learning on Protein Embeddings Enlightens Midnight Zone at Lightning Speed. *bioRxiv* **2021**. https://doi.org/10.1101/2021.11.14.468528.

(94) Stärk, H.; Dallago, C.; Heinzinger, M.; Rost, B. Light Attention Predicts Protein Location from the Language of Life. *Bioinforma. Adv.* **2021**, *1* (1), vbab035. https://doi.org/10.1093/bioadv/vbab035.

(95) Jumper, J.; Evans, R.; Pritzel, A.; Green, T.; Figurnov, M.; Ronneberger, O.; Tunyasuvunakool, K.; Bates, R.; Žídek, A.; Potapenko, A.; et al. Highly Accurate Protein Structure Prediction with AlphaFold. *Nature* **2021**, *596* (7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2.

(96) Senior, A. W.; Evans, R.; Jumper, J.; Kirkpatrick, J.; Sifre, L.; Green, T.; Qin, C.; Žídek, A.; Nelson, A. W. R.; Bridgland, A.; et al. Improved Protein Structure Prediction Using Potentials from Deep Learning. *Nature* **2020**, *577* (7792), 706–710. https://doi.org/10.1038/s41586-019-1923-7.

(97) Baek, M.; DiMaio, F.; Anishchenko, I.; Dauparas, J.; Ovchinnikov, S.; Lee, G. R.; Wang, J.; Cong, Q.; Kinch, L. N.; Schaeffer, R. D.; et al. Accurate Prediction of Protein Structures and Interactions Using a Three-Track Neural Network. *Science* **2021**, *373* (6557), 871–876. https://doi.org/10.1126/science.abj8754.

(98) Hie, B. L.; Yang, K. K.; Kim, P. S. Evolutionary Velocity with Protein Language Models Predicts Evolutionary Dynamics of Diverse Proteins. *Cell Syst.* **2022**, *13* (4), 274-285.e6. https://doi.org/10.1016/j.cels.2022.01.003.

(99) Ferruz, N.; Höcker, B. Towards Controllable Protein Design with Conditional Transformers. *arXiv* **2022**. https://doi.org/10.48550/arXiv.2201.07338.

(100) Yang, Y.; Arnold, F. H. Navigating the Unnatural Reaction Space: Directed Evolution of Heme Proteins for Selective Carbene and Nitrene Transfer. *Acc. Chem. Res.* **2021**, *54* (5), 1209–1225. https://doi.org/10.1021/acs.accounts.0c00591.

(101) Rix, G.; Watkins-Dulaney, E. J.; Almhjell, P. J.; Boville, C. E.; Arnold, F. H.; Liu, C. C. Scalable Continuous Evolution for the Generation of Diverse Enzyme Variants Encompassing Promiscuous Activities. *Nat. Commun.* **2020**, *11* (1), 5644. https://doi.org/10.1038/s41467-020-19539-6.

(102) Mazzaferro, C. Predicting Protein Binding Affinity With Word Embeddings and Recurrent Neural Networks. *bioRxiv* **2017**. https://doi.org/10.1101/128223.

(103) Nguyen, T.; Le, H.; Quinn, T. P.; Nguyen, T.; Le, T. D.; Venkatesh, S. GraphDTA: Predicting Drug–Target Binding Affinity with Graph Neural Networks. *Bioinformatics* **2021**, *37* (8), 1140–1147. https://doi.org/10.1093/bioinformatics/btaa921.

(104) Morrone, J. A.; Weber, J. K.; Huynh, T.; Luo, H.; Cornell, W. D. Combining Docking Pose Rank and Structure with Deep Learning Improves Protein–Ligand Binding Mode Prediction over a Baseline Docking Approach. *J. Chem. Inf. Model.* **2020**, *60* (9), 4170–4179. https://doi.org/10.1021/acs.jcim.9b00927.

(105) Ma, J.; Sheridan, R. P.; Liaw, A.; Dahl, G. E.; Svetnik, V. Deep Neural Nets as a Method for Quantitative Structure–Activity Relationships. *J. Chem. Inf. Model.* **2015**, *55* (2), 263–274. https://doi.org/10.1021/ci500747n.

(106) Ramsundar, B.; Kearnes, S.; Riley, P.; Webster, D.; Konerding, D.; Pande, V. Massively Multitask Networks for Drug Discovery. *arXiv* **2015**. https://doi.org/10.48550/arXiv.1502.02072.

(107) Šícho, M.; de Bruyn Kops, C.; Stork, C.; Svozil, D.; Kirchmair, J. FAME 2: Simple and Effective Machine Learning Model of Cytochrome P450 Regioselectivity. *J. Chem. Inf. Model.* **2017**, *57* (8), 1832–1846. https://doi.org/10.1021/acs.jcim.7b00250.

(108) Xiong, Y.; Qiao, Y.; Kihara, D.; Zhang, H.-Y.; Zhu, X.; Wei, D.-Q. Survey of Machine Learning Techniques for Prediction of the Isoform Specificity of Cytochrome P450 Substrates. *Curr. Drug Metab.* **2019**, *20* (3), 229–235. https://doi.org/10.2174/1389200219666181019094526.

(109) Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**, *115* (3), 211–252. https://doi.org/10.1007/s11263-015-0816-y.

(110) Li, M. M.; Huang, K.; Zitnik, M. Representation Learning for Networks in Biology and Medicine: Advancements, Challenges, and Opportunities. *arXiv* **2021**. https://doi.org/10.48550/arXiv.2104.04883.

(111) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The Protein Data Bank. *Nucleic Acids Res.* **2000**, *28* (1), 235–242. https://doi.org/10.1093/nar/28.1.235.

(112) Sillitoe, I.; Bordin, N.; Dawson, N.; Waman, V. P.; Ashford, P.; Scholes, H. M.; Pang, C. S. M.; Woodridge, L.; Rauer, C.; Sen, N.; et al. CATH: Increased Structural Coverage of Functional Space. *Nucleic Acids Res.* **2021**, *49* (D1), D266–D273. https://doi.org/10.1093/nar/gkaa1079.

(113) Wang, C. Y.; Chang, P. M.; Ary, M. L.; Allen, B. D.; Chica, R. A.; Mayo, S. L.; Olafson, B. D. ProtaBank: A Repository for Protein Design and Engineering Data. *Protein Sci.* **2018**, *27* (6), 1113–1124. https://doi.org/10.1002/pro.3406.

(114) Fowler, D. M.; Fields, S. Deep Mutational Scanning: A New Style of Protein Science. *Nat. Methods* **2014**, *11* (8), 801–807. https://doi.org/10.1038/nmeth.3027.

(115) Fowler, D. M.; Stephany, J. J.; Fields, S. Measuring the Activity of Protein Variants on a Large Scale Using Deep Mutational Scanning. *Nat. Protoc.* **2014**, *9* (9), 2267–2284. https://doi.org/10.1038/nprot.2014.153.

(116) Moult, J.; Fidelis, K.; Kryshtafovych, A.; Schwede, T.; Tramontano, A. Critical Assessment of Methods of Protein Structure Prediction (CASP) - Round X. *Proteins Struct. Funct. Bioinforma.* **2014**, *82* (SUPPL.2), 1–6. https://doi.org/10.1002/prot.24452.

(117) Ram, S.; Bepler, T. Few Shot Protein Generation. *arXiv* **2022**. https://doi.org/10.48550/arXiv.2204.01168.

(118) Dallago, C.; Mou, J.; Johnston, K. E.; Wittmann, B. J.; Bhattacharya, N.; Goldman, S.; Madani, A.; Yang, K. K. FLIP: Benchmark Tasks in Fitness Landscape Inference for Proteins. *bioRxiv* **2021**. https://doi.org/10.1101/2021.11.09.467890.

(119) Miton, C. M.; Tokuriki, N. How Mutational Epistasis Impairs Predictability in Protein Evolution and Design. *Protein Sci.* **2016**, *25* (7), 1260–1272. https://doi.org/10.1002/pro.2876.

(120) Starr, T. N.; Thornton, J. W. Exploring Protein Sequence–Function Landscapes. *Nat. Biotechnol.* **2017**, *35* (2), 125–126. https://doi.org/10.1038/nbt.3786.

(121) Kaznatcheev, A. Computational Complexity as an Ultimate Constraint on Evolution. *Genetics* **2019**, *212* (1), 245–265. https://doi.org/10.1534/genetics.119.302000.

(122) Bloom, J. D.; Silberg, J. J.; Wilke, C. O.; Drummond, D. A.; Adami, C.; Arnold, F. H. Thermodynamic Prediction of Protein Neutrality. *Proc. Natl. Acad. Sci.* **2005**, *102* (3), 606–611. https://doi.org/10.1073/pnas.0406744102.

(123) 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* **2016**, *5*, e16965. https://doi.org/10.7554/eLife.16965.

(124) Georgiev, A. G. Interpretable Numerical Descriptors of Amino Acid Space. *J. Comput. Biol.* **2009**, *16* (5), 703–723. https://doi.org/10.1089/cmb.2008.0173.

(125) Bai, S.; Kolter, J. Z.; Koltun, V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv* **2018**. https://doi.org/10.48550/arXiv.1803.01271.

(126) Chen, T.; Guestrin, C. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 2016; pp 785–794. https://doi.org/10.1145/2939672.2939785.

(127) Zhang, Y.; Zhou, X.; Cai, X. Predicting Gene Expression from DNA Sequence Using Residual Neural Network. *bioRxiv* **2020**. https://doi.org/10.1101/2020.06.21.163956.

(128) Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2018**. https://doi.org/10.48550/arXiv.1810.04805.

(129) Järvelin, K.; Kekäläinen, J. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* **2002**, *20* (4), 422–446. https://doi.org/10.1145/582415.582418.

(130) Bengio, Y.; Courville, A.; Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35* (8), 1798–1828. https://doi.org/10.1109/TPAMI.2013.50.

(131) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. *arXiv* **2017**. https://doi.org/10.48550/arXiv.1706.03762.

(132) Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9* (8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

(133) Yu, F.; Koltun, V.; Funkhouser, T. Dilated Residual Networks. *arXiv* **2017**. https://doi.org/10.48550/arXiv.1705.09914.

(134) Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models Are Few-Shot Learners. *arXiv* **2020**. https://doi.org/10.48550/arXiv.2005.14165.

(135) Steinegger, M.; Söding, J. Clustering Huge Protein Sequence Sets in Linear Time. *Nat. Commun.* **2018**, *9* (1), 2542. https://doi.org/10.1038/s41467-018-04964-5.

(136) Jiang, S.; Zavala, V. M. Convolutional Neural Nets in Chemical Engineering: Foundations, Computations, and Applications. *AIChE J.* **2021**, *67* (9), e17282. https://doi.org/10.1002/aic.17282.

(137) Rawat, W.; Wang, Z. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Comput.* **2017**, *29* (9), 2352–2449. https://doi.org/10.1162/neco_a_00990.

(138) Jaganathan, K.; Kyriazopoulou Panagiotopoulou, S.; McRae, J. F.; Darbandi, S. F.; Knowles, D.; Li, Y. I.; Kosmicki, J. A.; Arbelaez, J.; Cui, W.; Schwartz, G. B.; et al. Predicting Splicing from Primary Sequence with Deep Learning. *Cell* **2019**, *176* (3), 535-548.e24. https://doi.org/10.1016/j.cell.2018.12.015.

(139) Proutski, V.; Holmes, E. SWAN: Sliding Window Analysis of Nucleotide Sequence Variability. *Bioinformatics* **1998**, *14* (5), 467–468. https://doi.org/10.1093/bioinformatics/14.5.467.

(140) Tajima, F. Determination of Window Size for Analyzing DNA Sequences. *J. Mol. Evol.* **1991**, *33* (5), 470–473. https://doi.org/10.1007/BF02103140.

(141) Zhu, Z.; Wang, Y.; Zhou, X.; Yang, L.; Meng, G.; Zhang, Z. SWAV: A Web-Based Visualization Browser for Sliding Window Analysis. *Sci. Rep.* **2020**, *10* (1), 149. https://doi.org/10.1038/s41598-019-57038-x.

(142) Yang, Y.; Qian, W.; Zou, H. Insurance Premium Prediction via Gradient Tree-Boosted Tweedie Compound Poisson Models. *J. Bus. Econ. Stat.* **2018**, *36* (3), 456–470. https://doi.org/10.1080/07350015.2016.1200981.

(143) Zhou, H.; Qian, W.; Yang, Y. Tweedie Gradient Boosting for Extremely Unbalanced Zero-Inflated Data. *Commun. Stat. - Simul. Comput.* **2020**. https://doi.org/10.1080/03610918.2020.1772302.

(144) Ng, P. C. SIFT: Predicting Amino Acid Changes That Affect Protein Function. *Nucleic Acids Res.* **2003**, *31* (13), 3812–3814. https://doi.org/10.1093/nar/gkg509.

(145) Riesselman, A.; Shin, J.-E.; Kollasch, A.; McMahon, C.; Simon, E.; Sander, C.; Manglik, A.; Kruse, A.; Marks, D. Accelerating Protein Design Using Autoregressive Generative Models. *bioRxiv* **2019**. https://doi.org/10.1101/757252.

(146) Firnberg, E.; Labonte, J. W.; Gray, J. J.; Ostermeier, M. A Comprehensive, High-Resolution Map of a Gene's Fitness Landscape. *Mol. Biol. Evol.* **2014**, *31* (6), 1581–1592. https://doi.org/10.1093/molbev/msu081.

(147) Jacquier, H.; Birgy, A.; Le Nagard, H.; Mechulam, Y.; Schmitt, E.; Glodt, J.; Bercot, B.; Petit, E.; Poulain, J.; Barnaud, G.; et al. Capturing the Mutational Landscape of the Beta-Lactamase TEM-1. *Proc. Natl. Acad. Sci.* **2013**, *110* (32), 13067–13072. https://doi.org/10.1073/pnas.1215206110.

(148) Sarkisyan, K. S.; Bolotin, D. A.; Meer, M. V.; Usmanova, D. R.; Mishin, A. S.; Sharonov, G. V.; Ivankov, D. N.; Bozhanova, N. G.; Baranov, M. S.; Soylemez, O.; et al. Local Fitness Landscape of the Green Fluorescent Protein. *Nature* **2016**, *533* (7603), 397–401. https://doi.org/10.1038/nature17995.

(149) Sirin, S.; Apgar, J. R.; Bennett, E. M.; Keating, A. E. AB-Bind: Antibody Binding Mutational Database for Computational Affinity Predictions. *Protein Sci.* **2016**, *25* (2), 393–409. https://doi.org/10.1002/pro.2829.

(150) Yang, J.; Naik, N.; Patel, J. S.; Wylie, C. S.; Gu, W.; Huang, J.; Ytreberg, F. M.; Naik, M. T.; Weinreich, D. M.; Rubenstein, B. M. Predicting the Viability of Beta-Lactamase: How Folding and Binding Free Energies Correlate with Beta-Lactamase Fitness. *PLoS One* **2020**, *15* (5), e0233509. https://doi.org/10.1371/journal.pone.0233509.

(151) Henikoff, S.; Henikoff, J. G. Amino Acid Substitution Matrices from Protein Blocks. *Proc. Natl. Acad. Sci.* **1992**, *89* (22), 10915–10919. https://doi.org/10.1073/pnas.89.22.10915.

(152) Olson, C. A.; Wu, N. C.; Sun, R. A Comprehensive Biophysical Description of Pairwise Epistasis throughout an Entire Protein Domain. *Curr. Biol.* **2014**, *24* (22), 2643–2651. https://doi.org/10.1016/j.cub.2014.09.072.

(153) Nisthal, A.; Wang, C. Y.; Ary, M. L.; Mayo, S. L. Protein Stability Engineering Insights Revealed by Domain-Wide Comprehensive Mutagenesis. *Proc. Natl. Acad. Sci.* **2019**, *116* (33), 16367–16377. https://doi.org/10.1073/pnas.1903888116.

(154) Franks, W. T.; Wylie, B. J.; Stellfox, S. A.; Rienstra, C. M. Backbone Conformational Constraints in a Microcrystalline U- 15 N-Labeled Protein by 3D Dipolar-Shift Solid-State NMR Spectroscopy. *J. Am. Chem. Soc.* **2006**, *128* (10), 3154–3155. https://doi.org/10.1021/ja058292x.

(155) Livesey, B. J.; Marsh, J. A. Using Deep Mutational Scanning to Benchmark Variant Effect Predictors and Identify Disease Mutations. *Mol. Syst. Biol.* **2020**, *16* (7), e9380. https://doi.org/10.15252/msb.20199380.

(156) Shamsi, Z.; Chan, M.; Shukla, D. TLmutation: Predicting the Effects of Mutations Using Transfer Learning. *J. Phys. Chem. B* **2020**, *124* (19), 3845–3854. https://doi.org/10.1021/acs.jpcb.0c00197.

(157) Srinivas, N.; Krause, A.; Kakade, S. M.; Seeger, M. W. Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. *IEEE Trans. Inf. Theory* **2012**, *58* (5), 3250–3265. https://doi.org/10.1109/TIT.2011.2182033.

(158) Ebert, M. C.; Pelletier, J. N. Computational Tools for Enzyme Improvement: Why Everyone Can – and Should – Use Them. *Curr. Opin. Chem. Biol.* **2017**, *37*, 89–96. https://doi.org/10.1016/j.cbpa.2017.01.021.

(159) Lu, A. X.; Lu, A. X.; Moses, A. Evolution Is All You Need: Phylogenetic Augmentation for Contrastive Learning. *arXiv* **2020**. https://doi.org/10.48550/arXiv.2012.13475.

(160) Lu, A. X.; Zhang, H.; Ghassemi, M.; Moses, A. Self-Supervised Contrastive Learning of Protein Representations By Mutual Information Maximization. *bioRxiv* **2020**. https://doi.org/10.1101/2020.09.04.283929.

(161) Brandes, N.; Ofer, D.; Peleg, Y.; Rappoport, N.; Linial, M. ProteinBERT: A Universal Deep-Learning Model of Protein Sequence and Function. *Bioinformatics* **2022**, *38* (8), 2102–2110. https://doi.org/10.1093/bioinformatics/btac020.

(162) Shanehsazzadeh, A.; Belanger, D.; Dohan, D. Fixed-Length Protein Embeddings Using Contextual Lenses. *arXiv* **2020**. https://doi.org/10.48550/arXiv.2010.15065.

(163) Yang, J.; Anishchenko, I.; Park, H.; Peng, Z.; Ovchinnikov, S.; Baker, D. Improved Protein Structure Prediction Using Predicted Interresidue Orientations. *Proc. Natl. Acad. Sci.* **2020**, *117* (3), 1496–1503. https://doi.org/10.1073/pnas.1914677117.

(164) Odaibo, S. Tutorial: Deriving the Standard Variational Autoencoder (VAE) Loss Function. *arXiv* **2019**. https://doi.org/10.48550/arXiv.1907.08956.

(165) Rocklin, G. J.; Chidyausiku, T. M.; Goreshnik, I.; Ford, A.; Houliston, S.; Lemak, A.; Carter, L.; Ravichandran, R.; Mulligan, V. K.; Chevalier, A.; et al. Global Analysis of Protein Folding Using Massively Parallel Design, Synthesis, and Testing. *Science* **2017**, *357* (6347), 168–175. https://doi.org/10.1126/science.aan0693.

(166) Pokusaeva, V. O.; Usmanova, D. R.; Putintseva, E. V.; Espinar, L.; Sarkisyan, K. S.; Mishin, A. S.; Bogatyreva, N. S.; Ivankov, D. N.; Akopyan, A. V.; Avvakumov, S. Y.; et al. An Experimental Assay of the Interactions of Amino Acids from Orthologous Sequences Shaping a Complex Fitness Landscape. *PLOS Genet.* **2019**, *15* (4), e1008079. https://doi.org/10.1371/journal.pgen.1008079.

(167) Aakre, C. D.; Herrou, J.; Phung, T. N.; Perchuk, B. S.; Crosson, S.; Laub, M. T. Evolving New Protein-Protein Interaction Specificity through Promiscuous Intermediates. *Cell* **2015**, *163* (3), 594–606. https://doi.org/10.1016/j.cell.2015.09.055.

(168) Starita, L. M.; Pruneda, J. N.; Lo, R. S.; Fowler, D. M.; Kim, H. J.; Hiatt, J. B.; Shendure, J.; Brzovic, P. S.; Fields, S.; Klevit, R. E. Activity-Enhancing Mutations in an E3 Ubiquitin Ligase Identified by High-Throughput Mutagenesis. *Proc. Natl. Acad. Sci. U. S. A.* **2013**, *110* (14), E1263–E1272. https://doi.org/10.1073/pnas.1303309110.

(169) Fowler, D. M.; Araya, C. L.; Fleishman, S. J.; Kellogg, E. H.; Stephany, J. J.; Baker, D.; Fields, S. High-Resolution Mapping of Protein Sequence-Function Relationships. *Nat. Methods* **2010**, *7* (9), 741–746. https://doi.org/10.1038/nmeth.1492.

(170) Sgarbossa, D.; Lupo, U.; Bitbol, A.-F. Generative Power of a Protein Language Model Trained on Multiple Sequence Alignments. *arXiv* **2022**. https://doi.org/10.48550/arXiv.2204.07110.

(171) Podgornaia, A. I.; Laub, M. T. Pervasive Degeneracy and Epistasis in a Protein-Protein Interface. *Science* **2015**, *347* (6222), 673–677. https://doi.org/10.1126/science.1257360.

(172) Faure, A. J.; Domingo, J.; Schmiedel, J. M.; Hidalgo-Carcedo, C.; Diss, G.; Lehner, B. Mapping the Energetic and Allosteric Landscapes of Protein Binding Domains. *Nature* **2022**, *604* (7904), 175–183. https://doi.org/10.1038/s41586-022-04586-4.

(173) Gray, V. E.; Hause, R. J.; Luebeck, J.; Shendure, J.; Fowler, D. M. Quantitative Missense Variant Effect Prediction Using Large-Scale Mutagenesis Data. *Cell Syst.* **2018**, *6* (1), 116-124.e3. https://doi.org/10.1016/j.cels.2017.11.003.

(174) Smith, A. M.; Heisler, L. E.; St.Onge, R. P.; Farias-Hesson, E.; Wallace, I. M.; Bodeau, J.; Harris, A. N.; Perry, K. M.; Giaever, G.; Pourmand, N.; et al. Highly-Multiplexed Barcode Sequencing: An Efficient Method for Parallel Analysis of Pooled Samples. *Nucleic Acids Res.* **2010**, *38* (13), e142. https://doi.org/10.1093/nar/gkq368.

(175) Appel, M. J.; Longwell, S. A.; Morri, M.; Neff, N.; Herschlag, D.; Fordyce, P. M. UPIC–M: Efficient and Scalable Preparation of Clonal Single Mutant Libraries for High-Throughput Protein Biochemistry. *ACS Omega* **2021**, *6* (45), 30542–30554. https://doi.org/10.1021/acsomega.1c04180.

(176) Srivathsan, A.; Lee, L.; Katoh, K.; Hartop, E.; Kutty, S. N.; Wong, J.; Yeo, D.; Meier, R. ONTbarcoder and MinION Barcodes Aid Biodiversity Discovery and Identification by Everyone, for Everyone. *BMC Biol.* **2021**, *19* (1), 217. https://doi.org/10.1186/s12915-021-01141-x.

(177) Glenn, T. C.; Nilsen, R. A.; Kieran, T. J.; Sanders, J. G.; Bayona-Vásquez, N. J.; Finger, J. W.; Pierson, T. W.; Bentley, K. E.; Hoffberg, S. L.; Louha, S.; et al. Adapterama I: Universal Stubs and Primers for 384 Unique Dual-Indexed or 147,456 Combinatorially-Indexed Illumina Libraries (ITru & INext). *PeerJ* **2019**, *7*, e7755. https://doi.org/10.7717/peerj.7755.

(178) Wierbowski, S. D.; Vo, T. V; Falter-Braun, P.; Jobe, T. O.; Kruse, L. H.; Wei, X.; Liang, J.; Meyer, M. J.; Akturk, N.; Rivera-Erick, C. A.; et al. A Massively Parallel Barcoded Sequencing Pipeline Enables Generation of the First ORFeome and Interactome Map for Rice. *Proc. Natl. Acad. Sci.* **2020**, *117* (21), 11836–11842. https://doi.org/10.1073/pnas.1918068117.

(179) Chubiz, L. M.; Lee, M.-C.; Delaney, N. F.; Marx, C. J. FREQ-Seq: A Rapid, Cost-Effective, Sequencing-Based Method to Determine Allele Frequencies Directly from Mixed Populations. *PLoS One* **2012**, *7* (10), e47959. https://doi.org/10.1371/journal.pone.0047959.

(180) Weile, J.; Sun, S.; Cote, A. G.; Knapp, J.; Verby, M.; Mellor, J. C.; Wu, Y.; Pons, C.; Wong, C.; Lieshout, N.; et al. A Framework for Exhaustively Mapping Functional Missense Variants. *Mol. Syst. Biol.* **2017**, *13* (12), 957. https://doi.org/10.15252/msb.20177908.

(181) Campbell, N. R.; Harmon, S. A.; Narum, S. R. Genotyping-in-Thousands by Sequencing (GT-seq): A Cost Effective SNP Genotyping Method Based on Custom Amplicon Sequencing. *Mol. Ecol. Resour.* **2015**, *15* (4), 855–867. https://doi.org/10.1111/1755-0998.12357.

(182) Wen, C.; Wu, L.; Qin, Y.; Van Nostrand, J. D.; Ning, D.; Sun, B.; Xue, K.; Liu, F.; Deng, Y.; Liang, Y.; et al. Evaluation of the Reproducibility of Amplicon Sequencing with Illumina MiSeq Platform. *PLoS One* **2017**, *12* (4), e0176716. https://doi.org/10.1371/journal.pone.0176716.

(183) Siloto, R. M. P.; Weselake, R. J. Site Saturation Mutagenesis: Methods and Applications in Protein Engineering. *Biocatal. Agric. Biotechnol.* **2012**, *1* (3), 181–189. https://doi.org/10.1016/j.bcab.2012.03.010.

(184) Melnikov, A.; Rogov, P.; Wang, L.; Gnirke, A.; Mikkelsen, T. S. Comprehensive Mutational Scanning of a Kinase in Vivo Reveals Substrate-Dependent Fitness Landscapes. *Nucleic Acids Res.* **2014**, *42* (14), e112. https://doi.org/10.1093/nar/gku511.

(185) Sims, D.; Sudbery, I.; Ilott, N. E.; Heger, A.; Ponting, C. P. Sequencing Depth and Coverage: Key Considerations in Genomic Analyses. *Nat. Rev. Genet.* **2014**, *15* (2), 121–132. https://doi.org/10.1038/nrg3642.

(186) de Muinck, E. J.; Trosvik, P.; Gilfillan, G. D.; Hov, J. R.; Sundaram, A. Y. M. A Novel Ultra High-Throughput 16S RRNA Gene Amplicon Sequencing Library Preparation Method for the Illumina HiSeq Platform. *Microbiome* **2017**, *5* (1), 68. https://doi.org/10.1186/s40168-017-0279-1.

(187) Tresnak, D. T.; Hackel, B. J. Mining and Statistical Modeling of Natural and Variant Class IIa Bacteriocins Elucidate Activity and Selectivity Profiles across Species. *Appl. Environ. Microbiol.* **2020**, *86* (22). https://doi.org/10.1128/AEM.01646-20.

(188) Illumina. Nextera XT DNA Library Prep Reference Guide. 2019.

(189) Boville, C. E.; Romney, D. K.; Almhjell, P. J.; Sieben, M.; Arnold, F. H. Improved Synthesis of 4-Cyanotryptophan and Other Tryptophan Analogues in Aqueous Solvent Using Variants of TrpB from Thermotoga Maritima. *J. Org. Chem.* **2018**, *83* (14), 7447–7452. https://doi.org/10.1021/acs.joc.8b00517.

(190) Romney, D. K.; Murciano-Calles, J.; Wehrmüller, J. E.; Arnold, F. H. Unlocking Reactivity of TrpB: A General Biocatalytic Platform for Synthesis of Tryptophan Analogues. *J. Am. Chem. Soc.* **2017**, *139* (31), 10769–10776. https://doi.org/10.1021/jacs.7b05007.

(191) Buller, A. R.; Brinkmann-Chen, S.; Romney, D. K.; Herger, M.; Murciano-Calles, J.; Arnold, F. H. Directed Evolution of the Tryptophan Synthase β-Subunit for Stand-Alone Function Recapitulates Allosteric Activation. *Proc. Natl. Acad. Sci.* **2015**, *112* (47), 14599–14604. https://doi.org/10.1073/pnas.1516401112.

(192) Morrison, K. L.; Weiss, G. A. Combinatorial Alanine-Scanning. *Curr. Opin. Chem. Biol.* **2001**, *5* (3), 302–307. https://doi.org/10.1016/S1367-5931(00)00206-4.

(193) Alieva, A.; Aceves, A.; Song, J.; Mayo, S.; Yue, Y.; Chen, Y. Learning to Make Decisions via Submodular Regularization. *ICLR* **2021**.

(194) McInerney, P.; Adams, P.; Hadi, M. Z. Error Rate Comparison during Polymerase Chain Reaction by DNA Polymerase. *Mol. Biol. Int.* **2014**, *2014*. https://doi.org/10.1155/2014/287430.

(195) Hopf, T. A.; Green, A. G.; Schubert, B.; Mersmann, S.; Schärfe, C. P. I.; Ingraham, J. B.; Toth-Petroczy, A.; Brock, K.; Riesselman, A. J.; Palmedo, P.; et al. The EVcouplings Python Framework for Coevolutionary Sequence Analysis. *Bioinformatics* **2019**, *35* (9), 1582–1584. https://doi.org/10.1093/bioinformatics/bty862.

(196) Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; et al. API Design for Machine Learning Software: Experiences from the Scikit-Learn Project. *arXiv* **2013**. https://doi.org/10.48550/arXiv.1309.0238.

(197) Gibson, D. G.; Young, L.; Chuang, R.-Y.; Venter, J. C.; Hutchison, C. A.; Smith, H. O. Enzymatic Assembly of DNA Molecules up to Several Hundred Kilobases. *Nat. Methods* **2009**, *6* (5), 343–345. https://doi.org/10.1038/nmeth.1318.