Learning to Design and Engineer Proteins from Evolution, 3D Structures, and Experiments

Chloe Hsu

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-197
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-197.html

December 1, 2024

Learning to Design and Engineer Proteins from Evolution, 3D Structures, and Experiments

By

Chloe Hsu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jennifer Listgarten, Co-chair
Professor Moritz Hardt, Co-chair
Professor Yun S. Song
Professor David Schaffer

Spring 2023

Learning to Design and Engineer Proteins from Evolution, 3D Structures, and Experiments

Abstract

Learning to Design and Engineer Proteins from Evolution, 3D Structures, and Experiments

By

Chloe Hsu

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Jennifer Listgarten, Co-chair

Professor Moritz Hardt, Co-chair

Proteins serve many crucial functions in maintaining life, but have also been co-opted for human endeavors such as gene editing, immunotherapy, and plastic degradation. To better serve such needs, we re-engineer naturally existing proteins or design de novo proteins. In recent years, data at an unprecedented scale from evolution, 3D structures, and experiments became available for learning to design and engineer proteins. These distinct data types provide different yet complementary information about proteins. This thesis presents new machine learning methods that learn from multiple types of data for the problems of sequence-based protein fitness prediction and structure-based fixed backbone protein design.

For sequence-based protein fitness prediction, machine learning-based models typically learn from either unlabelled, evolutionarily-related sequences, or variant sequences with experimentally measured labels. For regimes where only limited experimental data are available, combining both sources of information could improve protein fitness prediction. Toward that goal, we propose a simple combination approach that is competitive with, and on average outperforms more sophisticated methods. The comparative analysis also highlights the importance of systematic evaluations and sufficient baselines.

For structure-based fixed backbone protein design, prior machine learning approaches to this problem have been limited by the number of available experimentally determined protein structures. We present a strategy to augment the training data by nearly three orders of magnitude by predicting millions of structures using AlphaFold2. Graph neural network and Transformer models trained with this additional data achieves an overall improvement of almost 10 percentage points over existing methods in native sequence recovery rate. We also study the generalization to a variety of more complex tasks including design of protein complexes, partially masked structures, binding interfaces, and multiple states.

To the wonders of biology and life.

Contents

List of Figures

List of Tables

Acknowledgments

Thank you to my advisors, Jennifer Listgarten and Moritz Hardt, for the support and encouragement over the years, especially for encouraging me to dive into the uncharted waters of a new interdisciplinary research area. Thank you Moritz for helping me see things as it is. Thank you Jennifer for helping me find the balance in tides and currents.

Thank you to Yun Song and David Schaffer for being on the committee. Thank you Yun for guiding me scientifically. Your curiosity and steadiness is a constant inspiration.

Thank you to Alex Rives, Adam Lerer, Tom Sercu, and Joshua Meier for taking me on the internship adventure. Thank you Adam for believing in me when I did not.

Thank you to the lovely group of "research-mates" at Berkeley—David Brookes, Akosua Busia, William DeWitt, Frances Ding, Clara Fannjiang, Milind Jagota, Hanlun Jiang, Maria Lukarska, Celestine Mendler-Dünner, Hunter Nisonoff, Samantha Robertson, Petr Skopintsev, Neil Thomas, Junhao Xiong, Brian Yao, and many more. I cherish each moment when our minds happen to flow through the same corner in science. Thank you Hunter, Akosua, Clara, and David for getting through the deep pandemic trenches together. Thank you Milind for together marveling at the magic of our immune systems.

Thank you to many other dear people who opened up my world in unexpected ways. Thank you also to the communities that gave me a home away from home at Berkeley.

Finally, thank you to my family for being a loving anchor in life.

The work in Chapter 2 was done together with Hunter Nisonoff, Clara Fannjiang, and Jennifer Listgarten. We thank A. Aghazadeh, P. Almhjell, F. Arnold, A. Busia, D. Brookes, M. Jagota, K. Johnston, L. Schaus, N. Thomas, Y. Wang, and B. Wittmann for helpful discussions. We would also like to thank P. Barrat-Charlaix, S. Biswas, J. Meier, and Z. Shamsi for providing helpful details about their methods and implementations.

Chapter 3 was based on joint work with Robert Verkuil, Jason Liu, Zeming Lin, Brian Hie, Tom Sercu, Adam Lerer, and Alexander Rives. We thank H. Akin, S. Candido, D. Ding, O. Kabeli, J. Meier, S. Ovchinnikov, P. Ramachandran, A. Rizvi, K. Wei, K. Yang, Z. Zhu, and the anonymous reviewers for feedback on the manuscript and insightful conversations.

Chapter 4 was written together with Clara Fannjiang and Jennifer Listgarten, with helpful feedback from Hanlun Jiang.

Chapter 1

Introduction

Naturally occurring proteins serve many crucial functions in maintaining life. They can be enzymes to catalyze essential chemical reactions, hormones to regulate various physiological processes, transporters to carry molecules across cell membranes, as well as structural components to support the shape of cells and tissues. Evolution creates new proteins primarily through the stochastic, generative mechanisms of mutation and recombination, coupled with the screening filter of natural selection.

Beyond naturally occurring proteins, proteins have also been co-opted for human endeavors such as gene editing [32, 58]; lighting up specific parts of cells [23]; therapeutic drugs [78]; and herbicide-resistant crops [97]. In many of these cases, we re-engineer proteins to better serve our needs. For example, we might enhance the original function, such as when we increase enzyme activity [69] or make green fluorescent proteins (GFPs) brighter [50]. Or, we might modify the original function to a related but different one, such as when we change an antibody to bind to a new target [16].

1.1 From Directed Evolution to Machine Learning-Guided Design

In the past few decades, new, useful, proteins have been engineered with directed evolution, which mimics natural generative mechanisms to create candidate amino acid sequences for user-specified selection. Figure 1.1 illustrates the directed evolution process. For a given protein function or property of interest, we typically start with a known protein sequence. This starting sequence is often known as the *wild-type* sequence and usually has a related function. From a starting sequence, the directed evolution approach then leverages laboratory-based generative mechanisms—such as error-prone PCR, site-saturated mutagenesis, and site-specific recombination, among others—to create a *library* of diversified protein sequence variants. This library of sequences is then experimentally screened for the function of interest, and subsequently the functionally enriched sequences among the screened candi-

dates are identified as an improved library. This library construction and screening process can be iterated for many rounds.

As DNA synthesis becomes increasingly scalable in the recent decades, correspondingly it is becoming feasible to replace the more constrained sequence construction approaches used in directed evolution with the most general mechanism: specifying entire protein sequences, amino-acid-by-amino-acid. To fully take advantage of the experimental capability to synthesize and screen large numbers of exact protein sequences, we need to more efficiently explore protein space *in silico* to find desired protein sequences to synthesize.

Prior to the recent adoption of machine learning for protein engineering and protein design, computational protein design typically relied on physics-based rational design [3, 72]. While researchers have successfully designed many *de novo* proteins with rational design, the computational methods for rational design can be computationally intensive and time-consuming and are limited by the accuracy of the energy functions used in the modeling process. Moreover, rational design often required significant domain knowledge about the specific protein families of interest.

Recently, machine learning points to a tantalizing new path for protein engineering and protein design. In the past two years, machine learning-based structure prediction methods [71, 11, 83, 143] have revolutionized structural biology; language models for protein sequences [105, 85] emerged with many downstream applications in protein design and in variant effect prediction; generative models for protein structures [140, 62, 5, 35] can in many cases generate feasible backbone structures based on functional sites or binding partners.

Figure 1.1: In the directed evolution approach, protein engineers typically generate a library of protein sequence variants related to a starting sequence, screen these variants for the desired properties or functions, and then iteratively construct new libraries for screening.

1.2 Learning from Evolution, 3D Structures, and Experiments

The rapid progress of machine learning in protein engineering and protein design is only possible with data becoming available at an unprecendented scale from evolution, 3D structures, and experiments. For example, structure prediction models such as AlphaFold not only learn from hundreds of thousands of 3D structures in the Protein Data Bank but also leverage evolutionary information from billions of metagenomic sequences. Among evolution, 3D structures, and experiments as three common data sources, they each have their own advantages and limitations. Evolutionary data are the most abundant. 3D structures are the most universal lens into protein functions. Experimental assays most directly get at each given function of interest. Rather than focusing on learning from a single data type, this thesis presents new methods that learn from multiple types of data.

In Chapter 2, we consider sequence-based protein fitness prediction, where fitness can refer to any protein property ranging from stability to enzyme activity and ligand binding. Protein fitness prediction models can be used to systematically screen all protein variants in a constrained design space; to search through vast design spaces in combination with an optimization algorithm; or to additionally filter candidate protein sequences generated by another design procedure. Machine learning-based models of protein fitness typically learn from either unlabeled, evolutionarily related sequences or variant sequences with experimentally measured labels. For regimes where only limited experimental data are available, we propose a simple approach to jointly learn from evolutionary data and experimental assay-labelled data to better inform protein fitness prediction models (Figure 1.2).

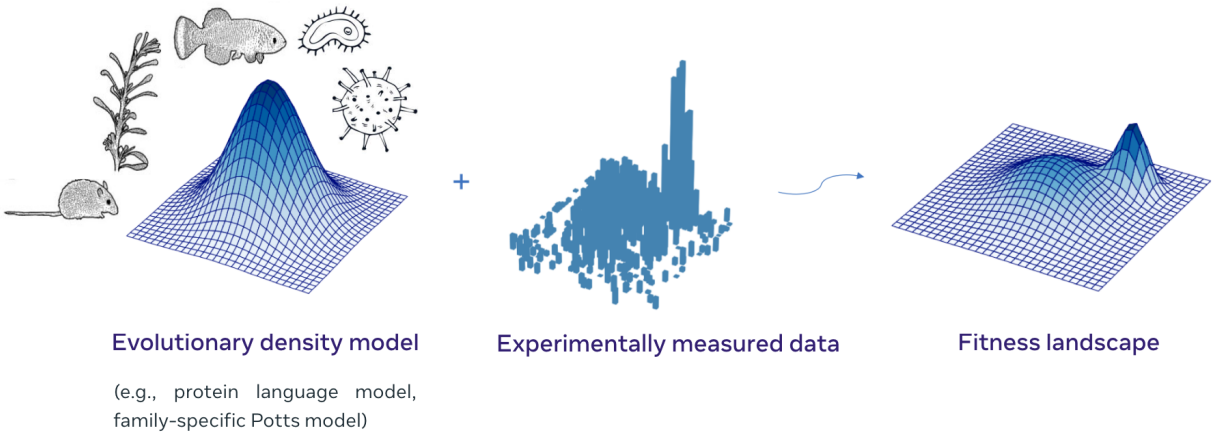

Figure 1.2: When experimental data are limited, combining evolutionary data and experimentally measured data improves protein fitness prediction.

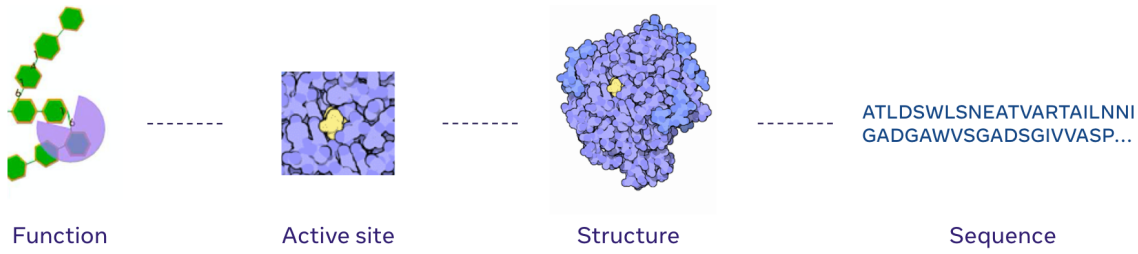

Figure 1.3: The 3D structures of proteins can be viewed as an intermediate representation between protein function and protein sequence.

In Chapter 3, we consider the structure-based protein design problem for a fixed backbone. Although in most applications ultimately we are interested in designing for protein functions, structure is nevertheless a very useful intermediate between sequence and function. Structures are particularly relevant for designing *de novo* proteins, as there are no similar naturally existing proteins as starting points for directed evolution. Today, to generate *de novo* protein designs with machine learning, researchers typically first generate the backbone structure and then generate sequences conditioned on the fixed backbone (Figure 1.3). We focus on the later step of generating sequences for a fixed backbone in Chapter 3. While previous fixed-backbone design methods typically only learn from experimentally determined 3D structures, we augment the existing set of experimentally determined 3D structures by generating orders of magnitude more predicted structures for evolutionary sequences.

In Chapter 4, we do not present any original research but comment on the potential of conditional generative models in protein design and protein engineering, intended for an audience less familiar with machine learning. The "inverse folding" model for fixed-backbone design in Chapter 3 is an example of a conditional generative model, which generates sequences conditioned on the backbone structure. We discuss what types of problems are well-suited to being tackled by conditional generative models, touch upon their training and sampling procedures, and introduce two instantiations of conditional generative models in protein engineering for sequence and structure generation respectively.

Looking forward, there are many other interesting data sources for learning important information about proteins. For example, transcriptomics can suggest the occurrence of proteins in their cellular contexts; proteomics can directly profile proteins along with their post-translational modifications; physical simulations and new imaging technologies can illustrate the conformational flexibility and the dynamics of proteins. Effectively learning from all these data sources will expand the reach of machine learning. Beyond proteins, now it is also an exciting time to envision truly general machine learning models for biology encompassing biochemistry, biophysics, cell biology, genetics, physiology, and more.

Chapter 2

Learning Protein Fitness Models from Evolutionary and Assay-labelled Data

Machine learning-based models that predict protein fitness from sequence can allow *in silico* screening to complement the existing protein engineering approaches of directed evolution and rational design. Here, fitness is a broad term that refers to any protein property ranging from stability to enzyme activity and ligand binding. In this chapter, we focus exclusively on machine learning (ML) methods for protein fitness prediction from protein sequences.

2.1 Background

Protein fitness models can be useful in many ways. Depending on the size of the design space and the computational cost of the protein fitness model, the model can be used either to systematically screen all protein variants in the design space [107, 109, 108, 141], or in combination with an optimization algorithm to search through vast design spaces [21, 20, 146, 120]. When in-depth knowledge of protein structure can be coupled to the phenotype of interest, physics-based methods such as FoldX [111], PoPMuSiC [30] and IMutant [22] can be used to model protein fitness. Applicable more generally, in the absence of such knowledge, machine learning models can learn from unlabelled, evolutionarily-related sequences, or from variant sequences with experimentally measured labels, to predict protein fitness.

There have been two main ML-based strategies for estimating protein fitness models. The first strategy uses implicit fitness constraints present in naturally occurring protein sequences, so-called *e*volutionary data. Such evolutionary methods start from one query protein with the desired property (*e. g.,* a particular GFP that fluoresces at some wavelength), and search through databases of naturally occurring proteins to find a set of related proteins—typically by sequence homology—that are assumed to be enriched for the same property as the query sequence. Then, a probability density model of this set of protein sequences is estimated; finally, sequence density evaluations are used to predict the relative fitness of protein variants of interest [53, 104]. Early methods for the related problem of

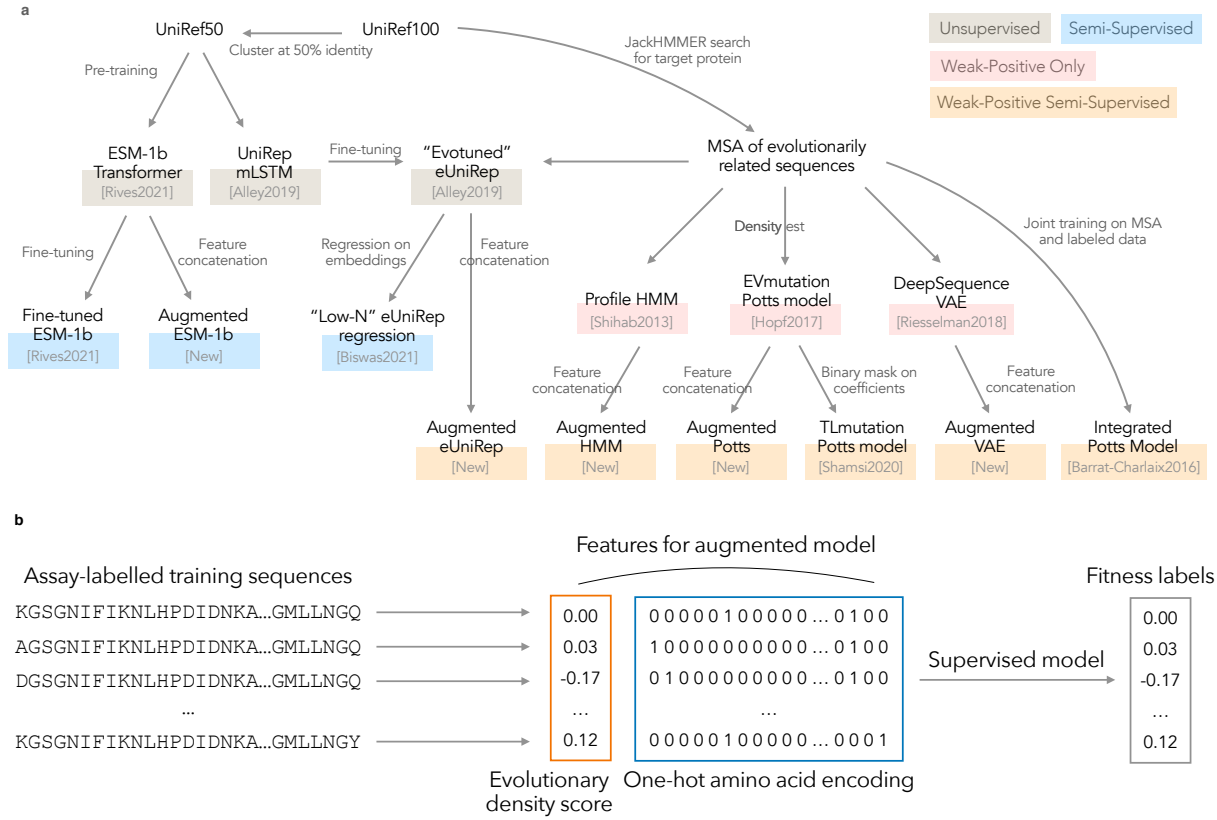

Figure 2.1: Machine learning methods for protein fitness prediction. (**a**) Overview of methods considered, divided into three learning strategies: (i) weak-positive-only learning methods which rely on probability density modelling of evolutionarily-related sequences (profile HMM [116], EVmutation [53] and DeepSequence [104]); (ii) semi-supervised learning methods which rely on unlabelled UniRef50 protein sequences, supervised data, and optionally evolutionarily-related sequences (fine-tuned ESM-1b [106] and eUniRep regression [17]); (iii) weak-positive semi-supervised learning methods which rely on evolutionaryily-related sequences and supervised data (the integrated Potts model [12], TLmutation [114], and augmented models). (**b**) Depiction of the augmented approach to predicting fitness wherein evolutionary and assay-labelled data are combined, and it is assumed that the probability density model has already been fully trained. In particular, the augmented approach uses linear regression with two kinds of features: first, evolutionary density evaluations (*i. e.*, the inferred sequence log-likelihoods or pseudo log-likelihoods from a sequence density model trained on evolutionarily related sequences), and, second, one-hot encoded site-specific amino acids, to perform supervised training on assay-labelled data.

pathogenicity prediction, such as SIFT [119] and Polyphen-2 [1], used position-specific substitution matrices (PSSMs), or Hidden Markov models [116]. In the context of protein fitness prediction, Potts models [87, 25, 41, 53, 109], which can model pairwise interactions in the sequence, have been shown to outperform SIFT and PolyPhen-2 at fitness prediction [53]. Deep-learning models such as variational autoencoders (VAEs) can capture higher-order interactions, and may provide more accurate predictions still [104]. Most evolutionary methods assume that the evolutionarily-related proteins have been aligned into a Multiple Sequence Alignment (MSA) [119, 1, 116, 87, 25, 41, 53, 109, 104]. Although the set of evolutionarily-related proteins doesn't typically have associated measurements for the property of interest, the homology search itself is assumed to implicitly provide a weak, positive labeling of all proteins in the set. Thus, we refer to these evolutionary methods as *w*eak-positive-only learning, rather than unsupervised learning. The number of sequences used for training in such cases can range from hundreds to hundreds of thousands when standard procedures are used to curate the set [53].

In the second main ML strategy for learning protein fitness models, supervised regression models are trained using variant sequences coupled with fitness labels measured in the laboratory. Depending on the protein and the property of interest, a supervised data set may comprise hundreds to hundreds of thousands of examples [10, 93]. The laboratory assay may be a relatively direct measurement of the fitness of interest [110], or a crude proxy [89]. Additionally, the set of variant sequences may be restricted to one or two mutations away from a query sequence [10, 89, 93], or be more heterogeneously distributed [110], with the former setting being more common. The labelled variant sequences may vary at only a few sequence positions (*e. g.,* four positions [142]), or vary more broadly [93]. By their limited construction, these data sets are not typically sufficiently rich to comprehensively characterize protein fitness landscapes, but are a good starting point. As time progresses, more comprehensive data sets, revealing increasingly more nuances of fitness landscapes, will expand our understanding. The simplest supervised approach is a linear regression model with one-hot encoded site-specific amino acid features; this approach can be extended to include pairwise features, and a non-linear transformation [96]. Richer supervised models used for protein fitness prediction include convolutional neural networks (CNNs) [141, 115], LSTMs [101, 4], and Transformers [101, 106, 84]. In some cases, such as that of LSTMs and Transformers, a large and broad set of unlabelled protein sequences is also used during training [4, 101, 106, 84, 17]; these data are thought to help find effective representations for the supervised learning component. Note that some of these classes of models, such as LSTMs and Transformers, can be employed either as probability density models or as supervised models.

Recently, several proposals have been made to combine these two ML strategies: using weak-positive-only learning on evolutionarily-related sequences together with supervised learning on assay-labelled variant sequences [17, 114, 12]. We refer to this setting as *w*eak-positive semi-supervised learning. For many scenarios of practical importance, property labels can only be assayed for hundreds of protein sequences. Especially in such a regime, but not only, it is important to combine both sources of data. Barrat-Charlaix *et al.* [12]

introduce an elegant approach to do so, the integrated Potts model, which assumes the Potts model energy function is identical to the supervised regression model; these are then trained jointly. Shamsi *et al.* [114] instead "transfer" information from a learned evolutionary model to a supervised version of the model by learning a binary masking of parameters. Motivated by the field of Natural Language Processing (NLP) [55, 31], Biswas *et al.* [17] first train an unsupervised autoregressive multiplicative LSTM (mLSTM) on a large database of naturally occurring protein sequences, then fine-tuning it with evolutionary data to produce a fixed-length latent representation called *e*UniRep. This representation is then used as the features in a regularized linear regression on the assay-labelled data.

In this chapter, we report on a systematic assessment of these "combined" methods as well as "pure" methods comprised within them, such as probability density-only models, or supervised-only models. We also introduce a simple baseline combined approach that turns out to be competitive with and often outperforms more sophisticated methods. In one instantiation, our approach achieves maximal performance on 15 of the 19 data sets used in our assessment—more than any other method—and is typically among the top competitors when it does not. Our simple approach can make use of any evolutionary probability density model, and adds little computational burden to it.

2.2 Results

Published machine learning methods assessed

We assessed a total of 13 published machine learning methods for protein fitness prediction that use evolutionary data, assay-labelled data, or both. Some [4, 106, 17] additionally make use of a large "universal" set of unlabelled protein sequences, namely the UniRef50 database [128]. In addition to the three combined methods summarized earlier (eUniRep regression, the integrated Potts model, and TLmutation), we also included methods that use only one or the other type of data so as to glean their relative importance across a range of settings. In particular, we included three representative purely evolutionary methods: a profile HMM [116], a Potts model (EVmutation [53]) and a VAE (DeepSequence [104]). We also included two supervised methods that do not use evolutionary data: a linear regression model with one-hot encoded site-specific amino acid features that uses only assay-labelled data, and the ESM-1b Transformer [106] model which was pre-trained on UniRef50 (unsupervised), and then fine-tuned with the assay-labelled data (supervised). While several Transformers have been developed and published for protein sequences [101, 37], we chose the ESM-1b model because at the time of our assessment, it was shown to have had superior performance on a number of prediction tasks when compared to other Transformers [106]. Figure 2.1a shows a graphical overview of all of these methods.

To enable all of these methods to run on our entire suite of benchmarking data sets we made minor modifications to some of the training procedures so that they could run in a reasonable amount of time. For each modification used, we show that the resulting

performance closely matches the performance of the original procedure on data sets used in the original papers (see Methods).

A simple baseline approach

We developed a simple approach that makes use of both evolutionary and assay-labelled data, and that turns out to be competitive with much more expensive and complicated approaches. For any already-trained evolutionary probability density model, we use ridge regression on one-hot encoded site-specific amino acid features augmented with one other feature—the sequence density evaluation (Figure 2.1b). We refer to this approach as "augmenting" a probability density model. For example, when the evolutionary probability density model is a Potts model, our baseline yields an *a*ugmented Potts model. Beyond augmented Potts models, we also compare to augmented VAEs and so forth. The augmented Potts model in particular can be interpreted in a Bayesian light as updating a prior from the evolutionary data, with assay-labelled data (see Methods). Because we use only site-specific amino acid features in the regression augmentation, the Bayesian update is focused on the site-specific parameters (as opposed to the pairwise coupling parameters). If we had included also pairwise amino acid features, then we would be more directly updating all parameters in the Potts model prior, which would yield an approach that is conceptually similar to the integrated Potts model [12], albeit with a different optimization objective. Given an already-trained probability density model, the augmentation entails only training a linear regression model, thus incurring little computational burden. Although incorporating richer evolutionary density-derived features, such as pairwise potentials, into the augmented regression could yield further improvements, we sought a baseline that did not require feature selection or specialized regularization[2]. We did, however, investigate extending our augmented regression features to include density evaluations from multiple models, such as from both a Potts model and a VAE, or from both a Potts model and a Transformer, and so forth. However, doing so resulted in only marginal gains on top of the augmented DeepSequence VAE model (data not shown). We speculate that this may be because the different probability density models were not providing sufficiently independent information.

Assay-labelled and evolutionary data sets

We assessed all methods on 19 labelled mutagenesis data sets, each comprising hundreds to tens of thousands of mutant sequences. Most related work [114, 106, 104, 12] evaluates on a subset of, or all of the mutation effect data sets in EVmutation [53]. We obtained our 19 data sets by including all EVmutation [53] protein data sets with at least 100 entries (in order to have sufficient data to glean insights from), and also included a GFP fluorescence data set [110] as in Biswas *et al.* [17] (however, their exact data set was not available at the time of our assessment). Other data sets that might have been relevant had insufficient evolutionary data in the MSAs [93, 142]. In each data set used, mutations were spread across either a domain, or the whole protein. Although most (16 out of the 19) labelled data

sets consisted only of sequences that were one mutation away from a fixed wild-type protein ("single mutants"), we also include detailed case studies of the three data sets [110, 89, 121] that contained sequences more than one mutation away from a wild-type protein ('higher-order mutants'). Table A.1 provides a detailed overview of these data sets. Each labelled data set was paired with an evolutionary data set found by searching through the UniRef100 database [128] for sequences similar to a single, relevant wild-type protein sequence, using Jackhmmer [42, 104, 53, 17]. For ease of comparison with other papers [114, 104, 53], when available, we directly used MSAs curated in this way and provided by EVmutation [53]. Otherwise, we used Jackhmmer with parameters set as in EVmutation.

Experimental overview

For each data set we always used 20% of the available labelled data for test sets (which by definition are not used to train or perform hyper-parameter selection). Given a fixed test set, we systematically varied the supervised training data set size by randomly sampling each of $48, 72, \cdots, 216, 240$ training examples from the non-test sequences (each training data set size was increased in intervals of 24). In addition to these fixed training data set sizes, for more direct comparison with TLmutation [114] and ESM-1b [106], we also used all remaining 80% of the available data not used for testing, which we refer to as an 80%-20% train-test split. When computationally feasible, we used 5-fold cross-validation on the training data set for hyper-parameter selection, and otherwise held out 20% of the training data as validation data. For each training data set size, including the the 80-20 split, we averaged performance over 20 random seeds that randomized the data partitions.

We used two measures of predictive model performance: (i) a Spearman rank correlation between the true and predicted fitness values [53, 104, 106, 12, 109], and (ii) a ranking measure from the information retrieval community called normalized discounted cumulative gain (NDCG) [64, 141], which gives high values when the top predicted results are enriched for truly high fitness proteins—that is, when the model is predicting the ranking well. In contrast to Spearman correlation, where errors at the top of the true ranked list carry the same weight as mismatches at the bottom, NDCG puts a higher weight on ranking the top of the list correctly and can be seen as a smoothed version of top-k average fitness without a fixed k. No one metric is necessarily the most relevant, as this will depend on the task at hand. For example, in using the model to propose a list of candidates for protein engineering, we may be interested only in the best performance among the top k proteins that will be assessed in the laboratory. In contrast, if using the model to understand the fitness landscape of a protein, we may be interested in more general accuracy. By using both Spearman correlation and NDCG, we hope to be broadly informative across a wide variety of settings. For each Spearman correlation result in the main text, the corresponding result with NDCG is found in the Extended Data and the Supplementary Information. Generally, the two metrics yielded similar qualitative conclusions.

Results for TLmutation [114] are omitted from the main figures for several reasons. First, TLmutation is conceptually similar to our augmented Potts model, but is restricted to only

allow for zeroing out of the evolutionary Potts model parameters by way of supervised learning. Second, TLmutation was much more computationally expensive than the augmented Potts model. And finally, in a set of exploratory experiments, TLmutation performed worse than our augmented Potts model (see Figure A.3).

Performance of existing methods and the augmented Potts model

When comparing existing methods to each other and to our augmented Potts model, averaged over all data sets, the augmented Potts model typically outperformed existing methods, including more sophisticated methods such as eUniRep regression [17] (Figure 2.2). The sole exception was that the purely evolutionarily-based VAE outperformed the augmented Potts model on the double mutant data in the low data regime. However, later we show that the augmented VAE outperforms the VAE, suggesting that either the Potts model is not sufficiently expressive in some cases, or that the VAE has a more suitable inductive bias, or both. As expected, with increasing labelled training data, the supervised methods increase in accuracy. When breaking down the average performance into individual data sets, the augmented Potts model is competitive on a majority of data sets (Figure 2.8 and Figure 2.9 and Figures A.1-A.2). Somewhat surprisingly, the extremely simple, supervised-only, linear regression on one-hot encoded amino acid features, is also among the top performers in the 80%-20% split setting. Due to limited availability of data sets that include double mutants, the reported double mutant performance is averaged over only the three relevant data sets and hence is heavily influenced by their characteristics. We refer the reader to Figure 2.5 for the double mutant performance breakdown.

Augmented models as a general, simple, and effective strategy

Next we also augmented the profile HMM, the DeepSequence VAE, the eUniRep mLSTM, and the ESM-1b Transformer, comparing them to each other and to the augmented Potts model. Note that the Transformer is the only method that does not make use of evolutionary data, but was pre-trained on UniRef50. Overall, we found that the augmented VAE achieved the highest average performance among the augmented models, with the augmented Potts model a close second (Figure 2.3). No matter which density model we augmented, the augmented model always outperformed the corresponding non-augmented existing method, regardless of the training data set size.

Among all approaches evaluated here, and earlier, the augmented DeepSequence VAE tended to be the best performer on most data sets (Figure 2.4b). It worked particularly well relative to others when predicting enzyme activity, achieving maximal performance in all 9 of those protein data sets (Figure 2.4b). For binding affinity, the different augmented models performed comparably (Figure 2.4b).

Generally all models that make use of evolutionary data achieved higher Spearman correlation with larger effective MSA size—the weighted number of sequences in MSA after accounting for sequence similarity with sample reweighting. Meanwhile, the relative rank-

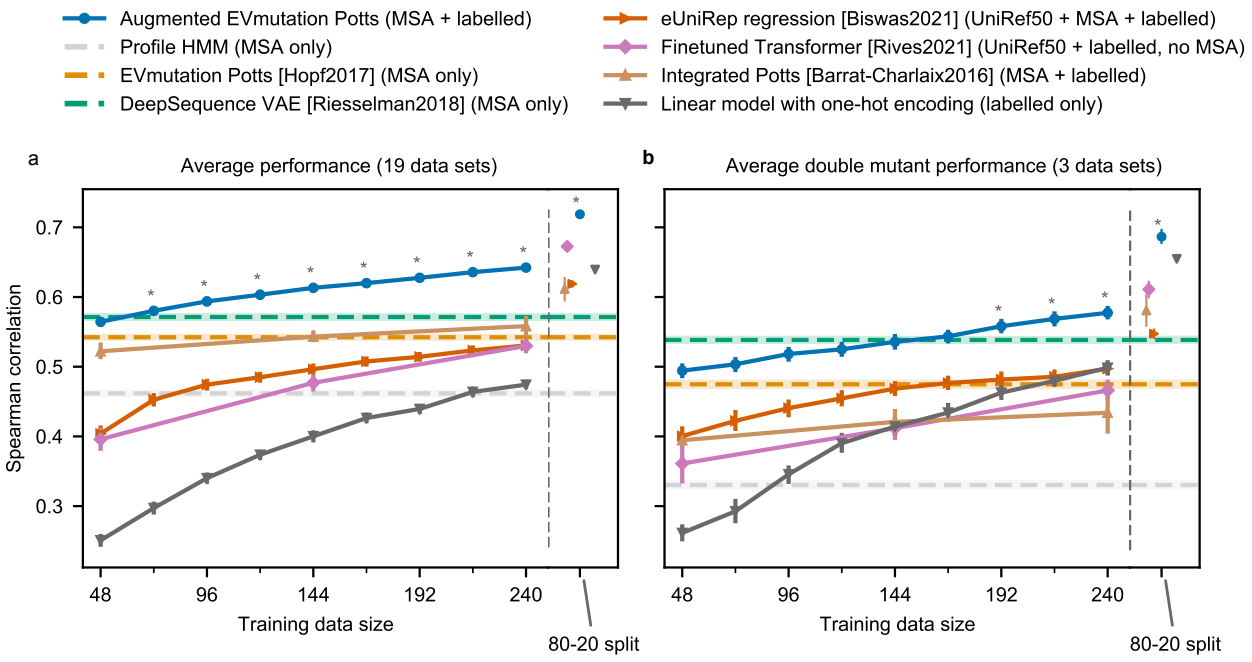

Figure 2.2: Performance of existing methods and the augmented Potts model. (**a**) Average performance across all 19 data sets, as measured by Spearman correlation. The horizontal axis shows the number of supervised training examples used. Error bars are centered at the mean and indicate bootstrapped 95% confidence intervals estimated from 20 random splits of training and test data. Note some bars are so small as to be almost invisible. Asterisks (*) indicate that $P < 0.01$ among all two-sided Mann-Whitney U tests that the augmented Potts model has different performance from each other method, at a given sample size. In particular, the largest such p-value for each training set size was respectively, $P = 9.0 \times 10^{-3}, 3.9 \times 10^{-7}, 9.2 \times 10^{-8}, 7.7 \times 10^{-4}, 6.8 \times 10^{-8}, 6.8 \times 10^{-8}, 6.8 \times 10^{-8}$ and 7.7×10^{-4} for training data size $72, 96, 120, \cdots, 240$ and $P = 7.7 \times 10^{-4}$ for the 80-20 split. (**b**) Average performance across all three data sets containing higher-order mutant sequences (sequences that are two mutations away from the wild-type), and restricted to testing on only double mutants. Asterisks (*) indicate same as in panel a), with largest p-values here, respectively, $P = 3.4 \times 10^{-4}, 2.7 \times 10^{-6}$ and 7.7×10^{-4} for training data size $192, 216, 240$ and $P = 7.7 \times 10^{-4}$ for the 80-20 split. See Figure 2.7 for NDCG and Figure 2.8 and Figure 2.9 and Figure A.1 and Figure A.2 for performance on individual data sets.

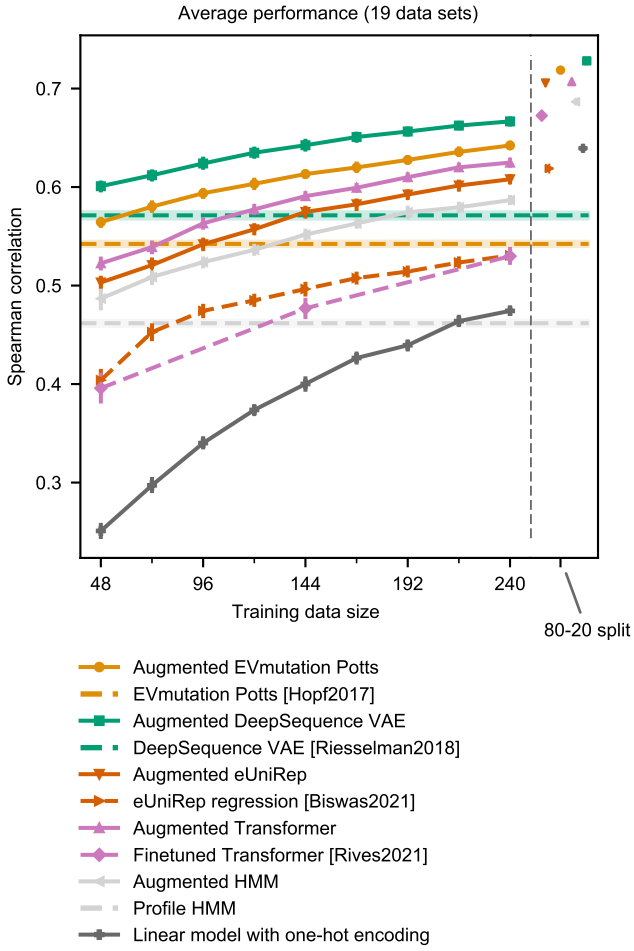

Figure 2.3: Augmented approach using different probability density models. Using the same evaluation setup as in Figure 2.2a, methods are compared with their augmented counterpart, using matching colors on each pair. Flat, horizontal lines represent evolutionary density models that do not have access to assay-labelled data. Dashed lines indicate previously-published methods. Error bars are centered at the mean and indicate bootstrapped 95% confidence interval from 20 random data splits. See Figure 2.10 for performance measured by NDCG.

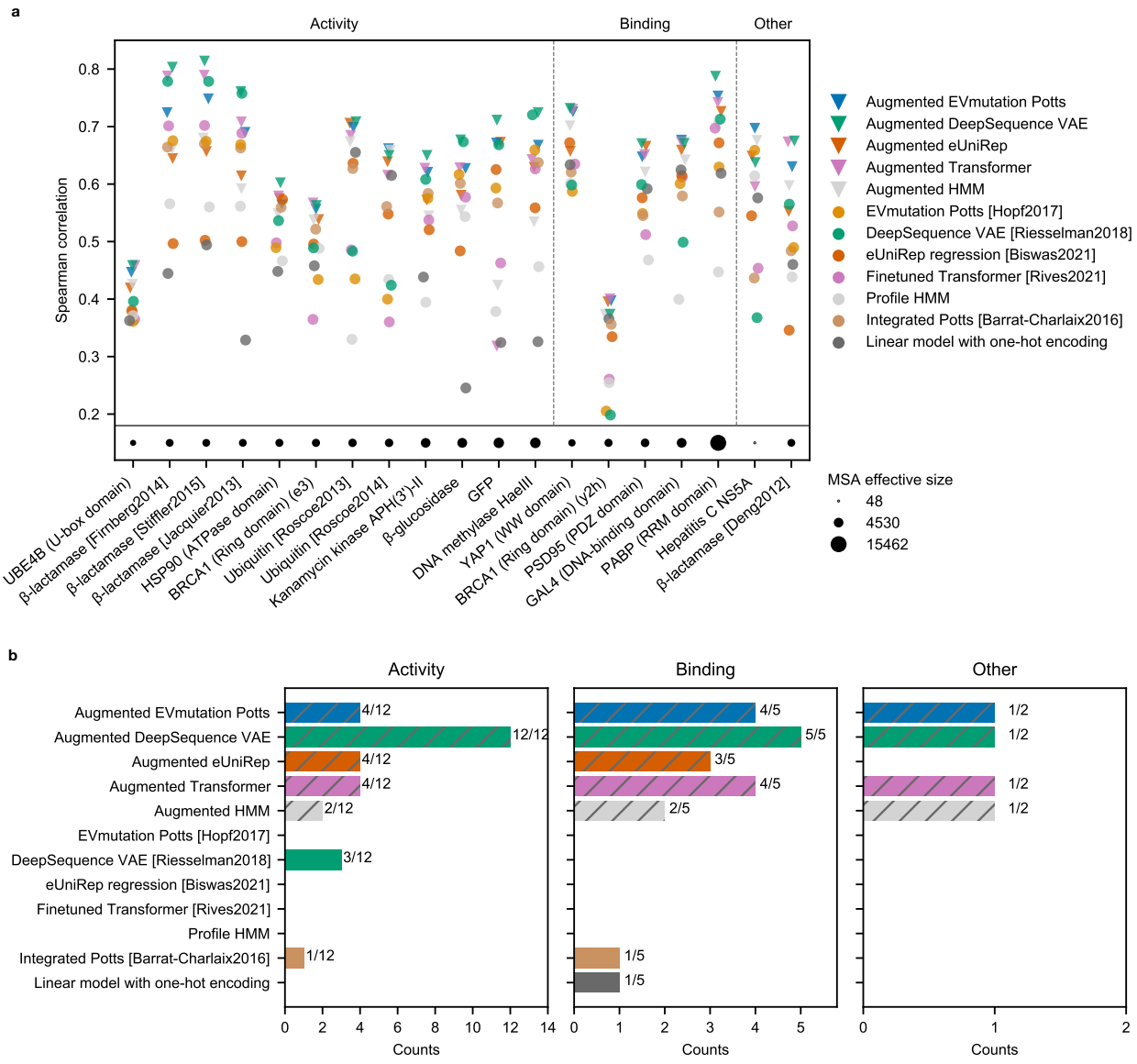

Figure 2.4: Performance on individual data sets. (**a**) Other than EVmutation, DeepSequence VAE, and Profile HMM, which do not use supervised data, all other methods here used 240 labelled training sequences. Each colored dot is the average Spearman correlation from 20 random train-test splits. Random horizontal jitter was added for display purposes. The bottom row of black dots indicates the effective MSA size after accounting for sequence similarity with sample reweighting at 80% identity cutoff. (**b**) Summary of how often each modelling strategy had higher Spearman correlation than all other strategies. Such modelling strategies were determined by first identifying the top-performing strategy for any given scenario, and then also identifying any other strategy that came within the 95% confidence interval of the top performer. See Figure 2.11 for analogous plots for NDCG and Figure 2.12 for evaluation on varying numbers of training examples.

ing of models appears to be unrelated to the effective MSA size (Figure 2.4a). Although each data set had a fixed MSA, we chose the data set with the largest effective MSA size (poly(A)-binding protein) to systematically reduce the size in order to observe the consequences of decreasing size. In this case study, we considered the augmented Potts model and linear regression (Figure A.4). The augmented Potts model performance degraded gradually when subsampling the MSA, while always outperforming linear regression, even with only 32 evolutionary sequences.

Extrapolation from single to higher-order mutants

Because 16 of the 19 supervised data sets consisted of only single mutants of a wild type, we next more closely examined the three data sets with higher-order mutants, namely, the green fluorescent protein (GFP) [110], Poly(A)-binding protein (PABP) RRM domain [89], and ubiquitination factor E4B (UBE4B) U-box domain [121] data sets. Earlier we divided each data set at random into train, validation and test, irrespective of the edit distance to wild-type. Here, however, we trained the models using only single-mutant labelled data, and then evaluated on single, double, triple, and quadruple mutants separately (Figure 2.5 and Figure A.5). We also tried training on both the single- and double-mutant data, and the results were qualitatively similar to using just the single mutant data for training, albeit with the relative difference in performance between any two models generally decreasing with more data (Figure 2.13 and Figure A.6). The extrapolative performance depends on how much epistasis contributes to the fitness landscape, and also on the ability of a model to capture such epistasis. The poor extrapolative performance on UBE4B may also stem from its evolutionary data not providing as much relevant information to the property of interest (*i. e.,* the assayed property).

In data sets where the assay-labelled variants comprise a variety of edit distances from the wild type, we find that simply using the edit distance itself is quite predictive of fitness relative to the ML models evaluated. For example, for GFP fluorescence, edit distance as a predictive model achieves a Spearman correlation of 0.45 (Figure 2.6), presumably because most mutations are deleterious and roughly additively so. This result suggests that correlation measures can be driven by simple features, even if captured by complex models. In contrast to GFP, mutation count is not as strongly predictive for the UBE4B U-box domain data (Figure 2.14), presumably because the mutational effects are more heterogeneous in the sense that they comprise either more diverse (both deleterious and beneficial), or non-additive effects, or both. Note that despite the bimodality of the experimentally measured fitness values, the predicted fitness values were unimodal from all five methods.

Adding in structure-based information

The augmentation approach can easily incorporate structure-based features or predictions from existing methods as additional regression inputs. Recent studies [43, 46, 61, 107, 141] have shown that protein structures, Rosetta energy terms, and molecular dynamics provide

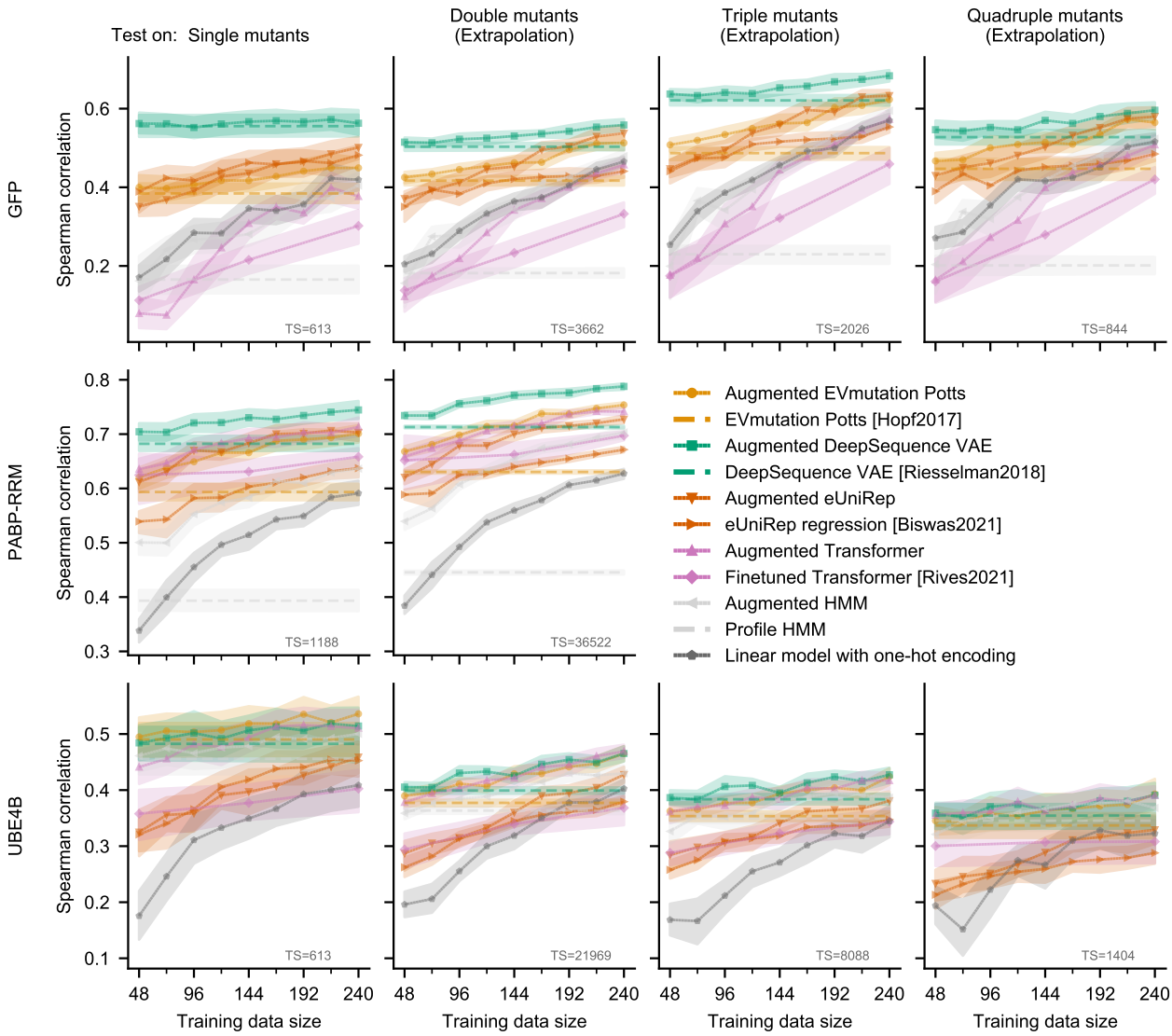

Figure 2.5: Extrapolative performance from single mutants to higher-order mutants. Each column shows the performance when training on randomly sampled single mutants and then separately testing on single, double, or triple mutants, none of which were in the training data. The "TS" value indicates the total number of mutants of a particular order in all of the data. For example, "TS=613" for single mutants means there were 613 total single mutants in the data set that we sampled from. Error bars are centered at the mean and indicate bootstrapped 95% confidence interval from 20 random data splits. Figure A.5 shows quantitatively similar results when measuring performance by NDCG, while Figure 2.13 and Figure A.6 show results for extrapolation when training on more than just single mutants (and therefore for larger training data set sizes).

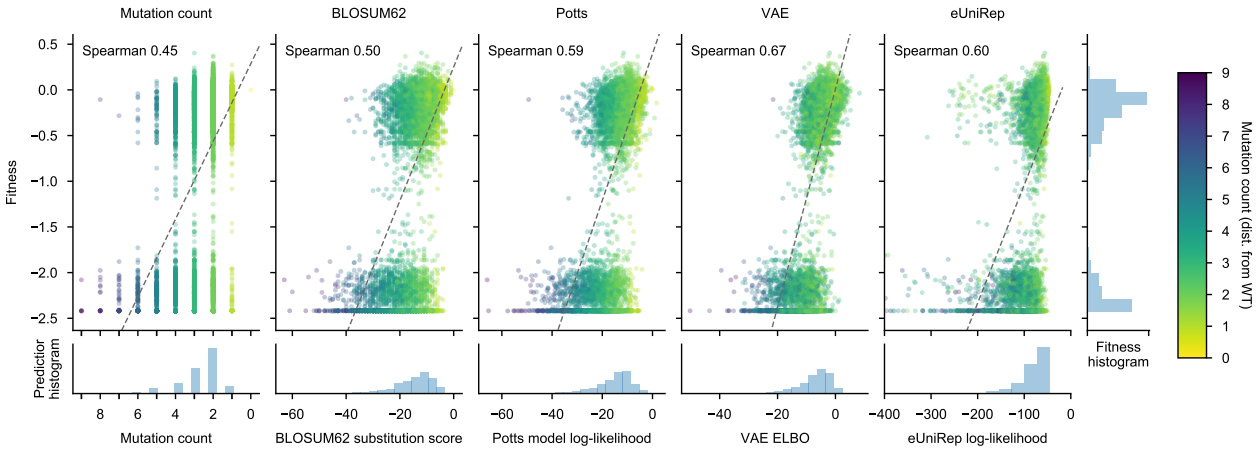

Figure 2.6: Sequence distance from wild-type sequence as a predictive model. On the GFP fluorescence data set, we compared the performance of non-augmented evolutionary density models to two predictive models that use only the distance of a sequence to the wild type. One is the edit distance, defined by the number of mutations away from the wild type. The other uses BLOSUM62 to compute the distance from wild type, which thus accounts not only for the number of mutations, but also the type of mutation. Each dot represents a GFP sequence, with darker colors indicating larger distances from the wild-type. See Figure 2.14 for the comparison on the UBE4B U-box domain data set.

valuable information for fitness prediction. A comprehensive study of how such features may improve fitness prediction is beyond the scope of the present work. However, to get an initial sense of where such future work may lead, we added FoldX-derived stability features to our models on the three data sets with higher-order mutants (Figure 2.15 and Figure A.7). The structure-based FoldX-derived features improved performance on GFP relative to the augmented VAE, but such improvements were not observed on PABP-RRM or UBE4B. These initial results suggest that whether structure-based features provide complementary information to the evolutionary and assay-labelled data may depend on the property of interest and on the quality of available structures.

2.3 Discussion

We compared ML-based methods that make use of both evolutionary and assay-labelled data for protein fitness prediction. In addition to comparing published methods, we introduced a simple baseline approach, wherein evolutionary density models are augmented with supervised data in a linear regression model on site-specific amino acid features. We instantiated this approach using density models in the form of a profile HMM, a Potts model, a VAE, an LSTM, and a Transformer. Across a wide range of settings, the augmented

Potts model was competitive with higher capacity and more computationally expensive approaches, such as the deep learning-based Transformer and LSTMs. Even in the relatively larger data regimes—which were still quite small—the extremely simple linear regression using site-specific one-hot encoded amino acid features performed quite well.

One may initially be puzzled as to how linear models with only site-specific features can generalize at all to mutations not seen at train time, let alone surpass non-linear models in this task. However, as detailed in the Methods, such generalization can emerge from a particular form of l_2-regularized linear regression with one-hot encoded features. In effect, through the regularization, the model learns about the importance of each *p*osition, even though each amino acid at each position has its own parameter. Thus if the effects of different mutations at the same position are in the same direction, the regularized linear models can do a reasonable job at generalizing in such a manner. Although higher-capacity deep learning-based approaches can in principle also pick up on this predictive power, they may be struggling to do so effectively, possibly owing to more degrees of freedom combined with a less suitable inductive bias. Having said that, deep learning models are not well-understood, with even the classical bias-variance trade-off now in question [48].

When comparing different augmented models, the augmented VAE was most effective at ranking mutational effects overall. Interestingly, even though the augmented ESM-1b Transformer does not use evolutionary data (but does use a large, "global" set of proteins for unsupervised pre-training), the augmented ESM-1b Transformer was competitive with methods that do have access to evolutionary data in the regime of relatively large training data sets. Thus, an augmented Transformer may be a promising choice for proteins with limited evolutionary data and a sufficiently large number of labelled training data. More accurate approximations to the Transformer sequence log-likelihood could improve performance further.

Approaches now considered standard in the mainstream ML community that use supervised data to fine-tune deep probability density models, such as Transformers and LSTMs, were herein outperformed by the augmented Transformers and LSTMs. These results highlight that state-of-the-art techniques from NLP should be carefully considered in the context of protein modelling. One substantial difference present in our setting is the existence of weak-positively labelled evolutionary data, which has no direct analogue in traditional NLP data sets.

As larger and more variable assay-labelled data sets emerge, we expect that increasingly richer models will dominate. Moreover, different machine learning strategies are likely needed for smaller and larger labelled data regimes, both of which are likely to remain relevant in protein engineering into the future. Therefore, when making general methodological recommendations, researchers should either consider implications across the full range of labelled data sizes or restrict recommendations to the regime tested.

To get an initial sense of how bringing in structure-based prediction might alter our narrative, we compared augmentation approaches with and without FoldX on three data sets, finding that FoldX-derived stability features could provide new, useful information. Going forward, it will be beneficial to examine this topic more comprehensively, including the use

of other tools such as Polyphen-2 [1], PoPMuSiC [30], IMutant [22], and AlphaFold2 [71], for example. As more features go into the augmentation, there are likely opportunities to use higher-capacity models for the augmentation model, beyond linear regression. These could be based on, for example, regression trees, or deep learning.

Although our goal was to assess different methods on real data rather than assuming certain simulation settings are representative, these real data sets have their limitations and are not necessarily representative of the test sets that one might encounter for protein engineering. In particular, in a protein engineering setting, one would like to understand how the model performs across a larger swath of protein space. Most importantly, one would like predictive models to extrapolate as accurately as possible to higher fitness areas than that observed in the data [40], and to be able to gauge when such extrapolations are unreliable [20]. While it may be possible to leverage techniques from covariate shift adaptation to help in this regard [127, 40], such approaches come with their own problems. We thus focused on directly evaluating methods without such corrections.

While the Spearman rank correlation has been used in previous studies [53, 104, 106, 114, 101], it does not capture all aspects of predictive performance. The Spearman rank correlation reflects a monotonic association between predicted and experimental mutational effects, but does not necessarily speak to anything about the mean squared error of those predictions, their scale, or any distributional characteristics of the fitness landscape, such as bimodality. However, as demonstrated by Wittman *et al.* [141], even modest Spearman correlation can be useful for speeding up iterative directed evolution. Our auxiliary use of NDCG, which does reflect aspects beyond the rank, provided added insight into the robustness of Spearman to assess predictive accuracy, as conclusions drawn from it were generally concordant to those based on the Spearman rank correlation. Ultimately, the specific task at hand will determine the suitability of any given metric, and any metric will have its shortcomings for a given task.

While we have focused on evaluating predictive performance, in cases where the predictive model is used in conjunction with a design strategy [146, 141, 120, 20, 40], one should additionally evaluate how these perform in tandem, in addition to independently.

2.4 Methods

Evolutionary sequences

We use the MSAs provided by EVmutation [53] whenever possible. For the green fluorescent protein (GFP), the only exception, we follow the same procedure as EVmutation to gather sequences using the profile HMM homology search tool Jackhmmer [42]. We determine the bit score threshold in Jackhmmer search with the same criterion from EVmutation. In particular, for GFP, we started with 0.5 bits/residue and subsequently lowered the threshold to 0.1 bits/residue to meet the sequence number requirement (redundancy-reduced number of sequences $\geq 10L$ where L is the length of the aligned region). For sensitivity analysis,

when using the bit score to 0.5 bits/residue or increasing the number of iterations to 10, the resulting MSAs on GFP still lead to similar downstream model performance (data not shown).

Mutation effect data sets

Hopf *et al.* [53] identified a list of mutation effect data sets generated by mutagenesis experiments of entire proteins, protein domains, and RNA molecules. We exclude the data sets for RNA molecules and influenza virus sequences, as well as excluding data sets that contain fewer than 100 entries, in order to have meaningful train/test splits with at least 20 examples in test data. This leaves us with 18 data sets from EVmutation.

Following the convention in EVmutation and DeepSequence, we exclude sequences with mutations at positions that have more than 30% gaps in MSAs, to focus on regions with sufficient evolutionary data. On most data sets, this excludes less than 10% of the data, although for a few proteins such as GFP this affects as much as half of the positions. For example, on GFP, out of 237 positions, only positions 15-150 pass the criterion of less than 30% gaps in the MSA. Coincidentally, the selected position 15-150 region covers the 81 amino region studied by Biswas *et al.* [17].

Among those 18 EVmutation data sets, only two on the poly(A)-binding protein activity [89] and the UBE4B auto-ubiquitination activity [121] include higher-order mutants.[†] While EVmutation only evaluates performance on the single mutants from the UBE4B U-box domain data [121], we also include higher-order mutants in evaluation. Additionally, we also include a green fluorescent protein (GFP) fluorescence data set [110], since GFP is a core example used by Biswas *et al.* [17]. The GFP data also include higher-order mutants. This results in 19 data sets total as listed in Table A.1.

Amino acid encodings

In addition to an overparameterized one-hot amino acid encoding (one binary feature for each amino acid possibility per position), we also experiment with the 19-dimensional physicochemical representation of the amino acid space developed by Georgiev [44], also used by Wittman *et al.* [141]. The features in Georgiev representation are principal components of over 500 amino acid indices from the AAIndex database [73]. As shown in Figure A.8, the Georgiev encoding achieves better performance than the one-hot amino acid encoding on data sets with higher-order mutants when presented with sufficient training data (bottom right), agreeing with previous findings [141]. However, on single-mutant only data sets or on limited training examples, the Georgiev encoding leads to almost identical performance as the one-hot encoding. For the augmented Potts model, the Georgiev encoding also does not improve performance compared to the one-hot encoding in any setting.

[†]Even though EVmutation Supplemental Table 1 indicates that the YAP1 (WW domain 1) peptide binding data set also include higher-order mutants, the supplemental data for YAP1 only contain single-mutant sequences, and no higher-order mutant data are available in the original publication [10] either.

Since the other evaluated methods, such as EVmutation [53], DeepSequence [104], and UniRep [4], all use the one-hot amino acid encoding as inputs to their models, we omit the Georgiev encoding results in the main text for fair comparison.

Linear model with one-hot encoded amino acid features

For a sequence $s = (s_1, \cdots, s_L)$ of length L, with elements $s_i \in \mathcal{A} = \{\text{all amino acids}\}$, the linear model with one-hot encoded amino acid features maps the sequence to a scalar regression output by

$$f(s;\theta) = \theta_0 + \sum_{i=1}^{L} \sum_{a \in \mathcal{A}} \mathbb{1}_a(s_i)\theta_i(a) = \theta_0 + \sum_{i=1}^{L} \theta_i(s_i), \tag{2.1}$$

where $\theta_0 \in \mathbb{R}$ is the bias term (which we deem not to be part of the encoding), $\mathbb{1}_a(s_i)$ is an indicator function equal to 1 when $a = s_i$ and to 0 otherwise, and each $\theta_i \in \mathbb{R}^{|\mathcal{A}|}$ is an $|\mathcal{A}|$-dimensional vector. Together, we refer to all of these parameters as θ. Each coefficient, $\theta_i(a) \in \mathbb{R}$, corresponds to the effect of a amino acid a at position i, and we use $\theta_i(s_i)$ to denote the coefficient corresponding to the particular amino acid s_i.

Importantly, this linear model is overparameterized by virtue of the one-hot encoding. In particular, there are $|\mathcal{A}| \times L + 1$ parameters for at most $(|\mathcal{A}| - 1) \times L + 1$ degrees of freedom. Additionally, if not all amino acids are observed at all positions in the training data, then the model will be correspondingly more overparameterized. As a consequence of this overparameterization, further constraints are needed to identify a unique solution. While there are different approaches for adding constraints, we chose to use ℓ_2-regularized regression (also known as ridge regression). We used five-fold cross-validation to determine the setting of the regularization parameter (λ below).

We deliberately chose to use this overparameterized approach over an approach with $|\mathcal{A}|-1$ non-redundant features, for reasons that will be explained shortly. Without regularization, this choice would be of no consequence. However, with regularization, the choice becomes consequential—both for the linear model with redundant one-hot encodings, and also for our augmented models which have a similar component. The main consequence of this choice is in how the trained model will make predictions on test sequences containing amino acids at positions that were not observed in the training data. Next we explain this in more detail.

Effects of ℓ_2 regularization on generalization

Minimizing the ℓ_2-regularized loss arising from the linear regression model in Equation 2.1 is equivalent to minimizing the following objective function,

$$g(\theta) := \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)};\theta))^2 + \lambda \sum_{i=1}^{L} \sum_{a \in \mathcal{A}} \theta_i(a)^2, \tag{2.2}$$

where the scalar λ is the regularization hyper-parameter that dictates the strength of the regularization, and $(s^{(n)}, y^{(n)})$ are the sequence-fitness pairs observed in the training data.

Generally we think of ℓ_2-regularized linear regression as tantamount to putting a spherical Gaussian prior on the regression parameters and obtaining a point estimate of the parameters, if one is a Bayesian. Alternatively, one may view it as a shrinkage penalty that pushes the weights towards zero. However, in the specific context herein—of an overparameterized model with one-hot encoded features—the ℓ_2 regularization has some further interesting properties that are not widely known. First, the use of any positive value for λ in the ℓ_2 regularization in this context implies that at any given position, i, the optimal parameters, $\hat{\theta}_i(a)$ (*i. e.,* those that minimize Equation 2.2), at that position, sum up to zero. We refer to this as the *z*ero-sum condition, which we now more formally define and prove, before discussing its consequences in more detail.

Lemma 1 (Zero-sum condition)**.** *Let $\hat{\theta}$ be the optimal parameters that minimize the ℓ_2-regularized linear regression objective (Equation 2.2) on a linear model with one-hot encoded amino acid features (Equation 2.1). For any position i, the coefficients over all the amino acids always sum to zero. That is,*

$$\sum_{a \in \mathcal{A}} \hat{\theta}_i(a) = 0.$$

Proof. By first order optimality conditions, at a solution to Equation 2.2, all of the partial derivatives of the ℓ_2-regularized objective (Equation 2.2) must all be equal to zero, namely,

$$\frac{\partial g(\theta)}{\partial \theta_0} = \left. \frac{\partial \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)}; \theta))^2}{\partial \theta_0} \right|_{\theta = \hat{\theta}} = 0, \text{ and} \tag{2.3}$$

$$\frac{\partial g(\theta)}{\partial \theta_i(a)} = \left. \frac{\partial \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)}; \theta))^2}{\partial \theta_i(a)} \right|_{\theta = \hat{\theta}} + 2\lambda \hat{\theta}_i(a) = 0 \text{ for all } i = 1, \ldots, L \text{ and all } a \in \mathcal{A}. \tag{2.4}$$

Furthermore, the sum of the indicator functions appearing in Equation 2.1, for position i, over all amino acids, is always equal to one,

$$\sum_{a \in \mathcal{A}} \mathbb{1}_a(s_i) = 1, \tag{2.5}$$

from which we see that

$$\sum_{a \in \mathcal{A}} \frac{\partial f(s; \theta)}{\partial \theta_i(a)} = \sum_{a \in \mathcal{A}} \frac{\partial (\theta_0 + \sum_{i=1}^{L} \theta_i(s_i))}{\partial \theta_i(a)} = \sum_{a \in \mathcal{A}} \mathbb{1}_a(s_i) = 1. \tag{2.6}$$

It also straightforwardly holds that the partial derivative of the model with respect to the bias is always equal to one, and therefore by the previous result is also equal to the sum of

partial derivatives,

$$\frac{\partial f(s;\theta)}{\partial \theta_0} = \frac{\partial(\theta_0 + \sum_{i=1}^{L} \theta_i(s_i))}{\partial \theta_0} = 1 = \sum_{a \in \mathcal{A}} \frac{\partial f(s;\theta)}{\partial \theta_i(a)}. \tag{2.7}$$

It therefore follows by two applications of the chain rule that for any value of θ,

$$\frac{\partial \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)};\theta))^2}{\partial \theta_0} = -2 \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)};\theta_0)) \frac{\partial f(s^{(n)};\theta)}{\partial \theta_0} \tag{2.8}$$

$$= -2 \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)};\theta)) \sum_{a \in \mathcal{A}} \frac{\partial f(s^{(n)};\theta)}{\partial \theta_i(a)} \tag{2.9}$$

$$= \sum_{a \in \mathcal{A}} -2 \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)};\theta)) \frac{\partial f(s^{(n)};\theta)}{\partial \theta_i(a)} \tag{2.10}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)};\theta))^2}{\partial \theta_i(a)}. \tag{2.11}$$

Now, more specifically at the optimal $\hat{\theta}$ that minimizes the objective, combining Equation 2.3 with Equation 2.11, we have

$$\sum_{a \in \mathcal{A}} \left. \frac{\partial \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)};\theta))^2}{\partial \theta_i(a)} \right|_{\theta=\hat{\theta}} = \left. \frac{\partial \sum_{n=1}^{N} (y^{(n)} - f(s^{(n)};\theta))^2}{\partial \theta_0} \right|_{\theta=\hat{\theta}} = 0, \tag{2.12}$$

which when combined with Equation 2.4, shows that

$$2\lambda \sum_{a \in \mathcal{A}} \hat{\theta}_i(a) = 0. \tag{2.13}$$

Therefore, so long as $\lambda > 0$, the ℓ_2-regularized solution must satisfy the zero-sum condition stated above, $\sum_{a \in \mathcal{A}} \hat{\theta}_i(a) = 0$. □

Generalization to mutations not seen at train time. If amino acid, α, at position i was not seen in the training data, then ridge regression sets $\hat{\theta}_i(\alpha) = 0$, so as to minimize the ℓ_2 penalty (Lemma 2, below). While we use a to generally denote amino acids, we use α to specifically refer to amino acids that are not seen at position i in the training data. A consequence of this property, jointly with the zero-sum condition, is that the effect, $\hat{\theta}_i(\alpha)$, of mutation α at position i on the predicted fitness, is precisely equal to the average effect of all amino acids seen at train time at that position, namely, $\hat{\theta}_i(\alpha) = \frac{1}{|\mathcal{A}_i^{train}|} \sum_{a \in \mathcal{A}_i^{train}} \hat{\theta}_i(a)$ (proof below), where $\mathcal{A}_i^{train}$ denote the set of all amino acids (including the wild type) seen at position i in the training data.

Consequently, the model will generalize to unseen amino acids at a given position in a manner that "understands" how mutable a site is, where mutability is the ability of the site to tolerate mutations while maintaining fitness. That is, if a given site tends to yield poor fitness when mutated, then its average training data effect will tend to be poor, and the extrapolated fitness at test time to a new amino acid will be predicted to be poor. To elucidate how much that information (mutability at a site) might perform on its own, we also tried a predictive model consisting of nothing but the position of the mutation (0/1) in a linear regression model (Figure 2.16 and Figures A.9-A.11). We found that this model is surprisingly predictive given that it is unaware of the specifics of any particular amino acids. In particular, it performs close to, but slightly worse than the linear model with one-hot encoded amino acid features. Notably, use of a non-redundant encoding in the same ℓ_2 modelling framework would not enable such predictions based on mutability to be made. In particular, in using one parameter per possible mutation from wild type at each position, the model would predict zero effect for mutations not seen at train time.

The mechanism for the site-specific generalization described above is similarly present in the augmented models because of their ℓ_2-regularized linear regression with one-hot encoded features, which retain an analogue to these properties even when augmented with a density feature. Note that one might consider altering the regularization to a more nuanced version wherein the emergent average amino acid prediction at a site is weighted in a manner to account for chemical similarity of amino acids, their frequency in the training data, and so forth.

Lemma 2 (Site-specific generalization effect)**.** *Let $\mathcal{A}_i^{train} := \{s_i^{(n)} \text{ for } s \in \mathcal{D}_{train}\}$ denote the set of all amino acids (including the wild type) seen at position i in the training data $\mathcal{D}_{train}$. Let $\alpha \notin \mathcal{A}_i^{train}$ be an amino acid at position i that was not seen in the training data. For the optimal parameters $\hat{\theta}$ that minimize the ridge regression objective, we must have $\hat{\theta}_i(\alpha) = 0$. Furthermore, $\hat{\theta}_i(\alpha)$ is the average effect of all seen amino acids at that position in the sense that $\hat{\theta}_i(\alpha) = \frac{1}{|\mathcal{A}_i^{train}|} \sum_{a \in \mathcal{A}_i^{train}} \hat{\theta}_i(a)$ and $\hat{\theta}_i(\alpha) - \theta_i(a_{WT}) = \frac{1}{|\mathcal{A}_i^{train}|} \sum_{a \in \mathcal{A}_i^{train}} (\hat{\theta}_i(a) - \theta_i(a_{WT}))$.*

Proof. Since $s_i^{(n)} \neq \alpha$ for any sequence $s^{(n)}$ in the training data, the first term in the objective function (Equation 2.2) does not depend on $\theta_i(\alpha)$. From the first-order optimality condition we therefore have

$$\left.\frac{\partial g(\theta)}{\partial \theta_i(\alpha)}\right|_{\theta=\hat{\theta}} = 2\lambda \hat{\theta}_i(\alpha) = 0 \implies \hat{\theta}_i(\alpha) = 0.$$

From the zero-sum condition,

$$\sum_{a \in \mathcal{A}_i^{train}} \hat{\theta}_i(a) + \sum_{a \notin \mathcal{A}_i^{train}} \hat{\theta}_i(a) = 0. \tag{2.14}$$

Since $\hat{\theta}_i(a) = 0$ for $a \notin \mathcal{A}_i^{train}$, we are then left with

$$\sum_{a \in \mathcal{A}_i^{train}} \hat{\theta}_i(a) = 0. \tag{2.15}$$

Therefore, for any $\alpha \notin \mathcal{A}_i^{train}$

$$\hat{\theta}_i(\alpha) = 0 = \frac{1}{|\mathcal{A}_i^{train}|} \sum_{a \in \mathcal{A}_i^{train}} \hat{\theta}_i(a). \tag{2.16}$$

If we prefer to consider instead the mutational effect $r$elative to wild-type, then we can obtain an analogous results simply by subtracting off $\theta_i(a_{WT})$ from both sides of the above equation (where a_{WT} denotes the wild-type amino acid), to get

$$\hat{\theta}_i(\alpha) - \theta_i(a_{WT}) = \frac{1}{|\mathcal{A}_i^{train}|} \sum_{a \in \mathcal{A}_i^{train}} (\hat{\theta}_i(a) - \theta_i(a_{WT})), \tag{2.17}$$

indicating that the results are true both for absolute and relative effects. □

EVmutation Potts model

Briefly, EVmutation learns a sequence distribution under site-specific constraints and pairwise constraints for each protein family. Under a Potts model (also sometimes known as a generalized Ising model) with alphabet $\mathcal{A}$, the probability of a sequence $s = (s_1, \cdots, s_L) \in \mathcal{A}^L$ of length L is given by a Boltzmann distribution:

$$P(s) = \frac{1}{\mathcal{Z}} \exp\{-\mathcal{H}(s)\}, \quad \mathcal{H}(s) = -\sum_{i<j}^{L} J_{ij}(s_i, s_j) - \sum_{i}^{L} h_i(s_i)$$

where $\mathcal{Z}$ is a normalization constant, $h_i \in \mathbb{R}^{|\mathcal{A}|}$ are site-specific amino-acid-specific parameters, $J_{ij} \in \mathbb{R}^{|\mathcal{A}| \times |\mathcal{A}|}$ are pair-specific amino-acid-specific parameters, and $s_i \in \mathcal{A}$ indicates the amino acid at position i. The h and J parameters are estimated by regularized maximum pseudo-likelihood. See EVmutation paper for the full modeling fitting details of sequence re-weighting, pseudolikelihood approximation, regularization, and optimization. We use the same plmc package (May 2018) as EVmutation to fit Potts models with default parameters.

Under the assumption that learned sequence probabilities are correlated with sequence fitness, EVmutation predicts the mutation effect of a mutant according to the log-odds ratios of sequence probabilities between the wild-type and mutant sequences, which is equivalent to the log-likelihood of the mutant up to a constant.

Profile HMM

Profile hidden Markov models (HMMs) [33] are probabilistic models that capture position-specific information about the amino acid distribution at each site, assuming that the amino acid at a particular position is independent of the amino acid at all other positions. Compared to Potts models, profile HMMs do not contain pairwise interactions, and are also referred to as the independent model or the independent site model. We evaluated the

HMMER suite profile HMM implementation [42] and the EVmutation independent model implementation [53] on the same evolutionary sequences. There is a minor performance difference (Figure A.12) between the two implementations due to algorithmic differences, and the "profile HMM" and "augmented HMM" methods in this chapter are based on the HMMER version.

DeepSequence VAE

Similar to EVmutation, DeepSequence [104] also models a sequence distribution for each protein family and predicts mutation effects according to approximations of log-odds ratios of sequence probabilities between the wild-type and mutant sequences. Here, the sequence distribution is modeled by a nonlinear variational autoencoder model with a multivariate Gaussian latent variable z. A neural network parameterizes the conditional distribution $p(s \mid z, \theta)$, and the sequence likelihood $p(s \mid \theta)$ is then

$$p(s \mid \theta) = \int p(s \mid z, \theta) p(z) dz. \tag{2.18}$$

While the exact log likelihoods are intractable, they are lower-bounded by the evidence lower bound (ELBO)

$$\mathcal{L}(\phi; s) = E_q[\log p(s \mid z, \theta)] - D_{KL}(q(z|s, \phi)||p(z)),$$

where $q(z \mid s, \phi)$ is a variational approximation for the posterior distribution $p(z \mid s, \theta)$ that is also modeled by a neural network. As an approximation to log-likelihoods, the ELBO can also be used to predict mutation effects. Whenever possible, we use the available ensemble predictions from DeepSequence [104]. For the GFP, the UBE4B U-box domain, and the BRCA1 ring domain, we followed the DeepSequence code in Theano and parameters to train an ensemble of VAE models (5 models with different random seeds). When computing the evidence-based lower bound (ELBO) as sequence log-likelihood estimations, we follow DeepSequence to take the average of 2000 ELBO samples (400 samples from each of the 5 VAE models in the ensemble).

eUniRep mLSTM

The UniRep [4] multiplicative long short-term memory network (mLSTM) can also be viewed as a sequence distribution. Given the first i amino acids of a sequence, a neural network parameterizes the conditional probabilities of the next amino acid $P(s_{i+1} \mid s_1 \cdots s_i)$. From the conditional probabilities we can also reconstruct the sequence probability as

$$P(s) = \Pi_{i=1}^{L} P(s_i \mid s_1 \cdots s_{i-1}).$$

For convenience, artificial start and stop tokens are added at the two ends of sequences. Under this probabilistic interpretation, the unsupervised pre-training on UniRef50 sequences with next-step predictions is equivalent to finding model parameters that maximize likelihood

on UniRef50 sequences. We omit the performance of the pre-trained (not evo-tuned) UniRep model in our results since generally it is largely outperformed by the eUniRep described below.

To adapt the UniRep model to specific protein families, Biswas *et al.* [17] propose the "evo-tuning" procedure, which fine-tunes the UniRep model by minimizing the same next-step prediction loss on evolutionary sequences in addition to on UniRef50 sequences. Following the naming convention [17], We refer to the model after "evo-tuning" as eUniRep. After fine-tuning, Biswas *et al.* [17] then perform supervised regression with the 1900-dimensional average embeddings (averaged over sequence length axis) from the final hidden layer in the eUniRep model as regression features.

While the original fine-tuning procedure for eUniRep [17] optimizes for next-step prediction loss on entire evolutionary sequences up to length 500 (discarding longer sequences), we find that next-step prediction on aligned portions of evolutionary sequences (with gaps included as gap tokens) works as well on GFP and beta-lactamase (Figure A.13), and therefore fine-tune on only aligned portions to lower computational costs as the mLSTM memory usage scales quadratically with respect to sequence length. We used the open-source UniRep code in tensorflow.

For each protein family, we randomly split the evolutionary sequences from the MSA into an 80% training set and a 20% validation set to check for over-fitting. When training for 10,000 gradient steps with the same learning rate 1×10^{-5} as Biswas *et al.* [17], the validation loss eventually plateaus but does not increase. Since the validation loss typically plateaus before 10,000 steps, we chose to stop training at 10,000 steps.

ESM-1b Transformer

The ESM-1b Transformer [106] is pre-trained on UniRef50 representative sequences with the masked language modeling objective, where a fraction (15%) of amino acids in each input sequence are masked and the model is trained to predict the missing tokens. We chose the ESM-1b model to represent Transformers as it has the best performance on downstream secondary structure prediction and contact prediction tasks [106] among Transformers from prior works [101, 37] and other Transformer architectures tested. While the original mutation effect prediction results [106] were based on the older 34-layer ESM-1 model, we found that ESM-1b slightly improves performance over the ESM-1 model (data not shown).

When using Transformer models for sequence fitness predictions, Rives *et al.* [106] mask the mutated positions and used the difference in conditional log-likelihoods (conditioned on non-mutated amino acids) between the mutated amino acids and the wild-type amino acids as fitness prediction. We explain below that this could be viewed as an approximation for pseudo log-likelihoods (PLLs).

Masked token language models can also be viewed as sequence distributions, where the sequence distribution is implicitly represented by conditional likelihoods $P(s_i \mid s_{-i})$, where s_{-i} indicates all other sequence positions excluding position i, i.e. $s_{-i} = s_1 \cdots s_{i-1} s_{i+1} \cdots s_L$.

Since exact likelihood are too computationally expensive for Transformers, we resort to pseudo-likelihoods. In general, given a sequence s of length L, its pseudo-likelihood [15] is defined as the product of conditional likelihoods for each site. Hence, the pseudo-log-likelihood of a sequence s is

$$PLL(s) = \sum_{i=1}^{L} \log P(s_i \mid s_{-i}).$$

However, even the evaluation of pseudo log-likelihoods (PLLs) is computationally expensive, since it requires L inferences for a sequence of length L. As a more computationally efficient approximation, we only compute conditional likelihoods on the mutated positions, and then use the difference between the conditional log-likelihoods of the mutated sequence and the wild-type sequence as an approximation for pseudo-log-likelihoods. More specifically, for a mutant sequence ϕ and wild-type sequence σ, we approximate the PLL difference by the following, as equivalent to the existing formula used by Rives *et al.* [106]:

$$\begin{aligned} PLL(\phi) - PLL(\sigma) &= \sum_{i=1}^{\ell} \log p(\phi_i \mid \phi_{-i}) - \sum_{i=1}^{\ell} \log p(\sigma_i \mid \sigma_{-i}) \\ &= \sum_{i:\phi_i \neq \sigma_i} (\log p(\phi_i \mid \phi_{-i}) - \log p(\sigma_i \mid \sigma_{-i})) + \underbrace{\sum_{i:\phi_i = \sigma_i} (\log p(\sigma_i \mid \phi_{-i}) - \log p(\sigma_i \mid \sigma_{-i}))}_{\approx 0} \end{aligned}$$

The underlying assumption here is that the conditional log-likelihoods of wild-type amino acids on mutant backgrounds are roughly the same as on wild-type backgrounds. While this might be accurate for mutant sequences that are close enough to wild-type sequences, in general there is no support for this approximation on high-order mutant sequences and the full pseudo-log-likelihoods will likely be more accurate for mutation effect predictions.

For supervised learning, we evaluated two approaches with the same Transformer model. The first one ("fine-tuned Transformer") is to fine-tune the entire Transformer model with fitness labels as done by Rives *et al.* [106], using the PLL difference between the mutant and the wild-type as a predictor for fitness. We perform 20 epochs of supervised fine-tuning with learning rate 3×10^{-5} and with early stopping according to validation Spearman correlation, using the open-source ESM code in PyTorch. For a given training data size, we keep 20% of the data for validation (early-stopping) and use the remaining 80% for fine-tuning. Using Spearman correlation as opposed to validation loss for early stopping is crucial for performance, especially on small data sizes. Although the implementation details might not exactly match those from Rives *et al.* (since the fine-tuning code is not available), we show that our method is able to reproduce the same results on most of the Envision data sets used by Rives *et al.*, as shown in Figure A.14.

In the second approach ("augmented Transformer"), while keeping the Transformer model constant, we concatenate the PLL difference inferred from the pre-trained (not fine-tuned) model together with one-hot amino acid encoding as features for regression.

We also attempted unsupervised fine-tuning ("evo-tuning") of the ESM-1b Transformer model on evolutionarily related sequences from MSAs, although our preliminary efforts on β-lactamase and PABP-RRM do not result in improved performance (data not shown).

Integrated (tied-energy) Potts model

The integrative approach [12] for Potts models optimizes the joint log-likelihood for model parameters on both evolutionary data and labelled data. Given evolutionary sequences $\sigma_1, \cdots, \sigma_M$, the log-likelihood on evolutionary data is

$$\log P(\sigma^1, \cdots, \sigma^M \mid J, h) = -\sum_{k=1}^{M} \mathcal{H}(\sigma^k) - M \log \mathcal{Z},$$

where

$$\mathcal{H}(\sigma) = -\sum_{i<j}^{L} J_{ij}(\sigma_i, \sigma_j) - \sum_{i}^{L} h_i(\sigma_i).$$

On the other hand, given sequence-fitness pairs $(s_1, y_1), \cdots, (s_N, y_N)$, assuming i.i.d. Gaussian noise drawn from $\mathcal{N}(0, \Delta^2)$, the log-likelihood on labelled data is

$$\log P(s^1, \cdots, s^M, y^1, \cdots, y^M \mid J, h) = -\frac{1}{2\Delta^2} \sum_{k=1}^{N} [y^k - \mathcal{H}(s^k)]^2 - \frac{N}{2} \log(2\pi\Delta^2).$$

The integrated Potts model is learned by maximizing the joint log-likelihood

$$\log P(\sigma^1, \cdots, \sigma^M \mid J, h) + \log P(s^1, \cdots, s^M, y^1, \cdots, y^M \mid J, h)$$

with ℓ_2-regularization.

The noise variance Δ^2 determines the relative weighting between the two losses in the joint log-likelihood. Using existing notation [12], the relative weighting parameter is

$$\lambda = \frac{1}{1 + \Delta^2}.$$

In practice, since we do not know the noise variance, we use 20% of the training data for validation and choose the best λ according to Spearman correlation on validation set. Another practical complication is that we only have fitness labels that are up to some monotonic transformation from the energies. Following Figliuzzi *et al.* [41], we use a monotonic mapping between sorted energy values and sorted fitness labels.

Following recommendations from the authors, the Potts model parameters are initialized with parameters estimated by pseudo-likelihood. Before introducing labelled data, we first give the model a warm start by training 500 iterations only on evolutionary data. Then, we optimize the regularized joint log-likelihood for 100 iteration for each λ value. The only

exception is for β-glucosidase, where the protein sequences are too long (> 500 amino acids) and lead to very high memory consumption and long run-time (> 2 days) for the integrated Potts model. Therefore, for β-glucosidase alone we use the Potts model performance without labelled data as a substitute for the integrated model performance.

We follow the publicly available code (https://github.com/PierreBarrat/DCATools/tree/master/src) with slight modifications to use zero-sum gauge instead of wild-type gauge for Potts models. In Potts models for categorical variables, there are more free parameters than independent constraints, and gauge fixing refers to reducing the number of independent parameters to match match the number of independent constraints [123]. The wild-type gauge forces all parameters corresponding to wild-type amino acids to be zero, while the zero-sum gauge requires $\sum_{\omega \in \mathcal{A}} h_i(\omega) = 0$ and $\sum_{\omega \in \mathcal{A}} J_{ij}(\omega, \alpha) = \sum_{\omega \in \mathcal{A}} J_{ij}(\alpha, \omega) = 0$. When there is regularization involved, different gauge fixings lead to non-equivalent models. Although gradient calculations are easier in the wild-type gauge, we found that the zero-sum gauge led to better predictive performance (Figure A.15) and hence adopted the zero-sum gauge.

Alternative tied-energy models

In addition to integrated Potts models, we also evaluate alternative ways to train a density model of a protein family and a predictive model of fitness in an integrated fashion (Figure A.16). Rather than using a fixed and possibly incorrect monotonic mapping between sorted energy values and sorted fitness labels [41], we consider a differentiable proxy of the Spearman correlation [18] between the predicted and true fitness values. Specifically, given N sequence-fitness pairs, $(s_1, y_1), \cdots, (s_N, y_N)$, let $(s_1, r(y_1)), \cdots, (s_N, r(y_N))$ denote the corresponding sequence-ranking pairs, where $r(y_k) \in \{1, \ldots, N\}$ gives the rank in descending order of the value y_k among the values $\{y_1, \ldots, y_N\}$. We fit a Potts model by optimizing the joint loss

$$\frac{1}{M} \log P(\sigma^1, \cdots, \sigma^M \mid J, h) + \lambda \frac{1}{N} \sum_{k=1}^{N} [r(y^k) - \hat{r}(\mathcal{H}(s^k))]^2$$

with ℓ_2-regularization, where $\hat{r}(\mathcal{H}(s^k))$ is a differentiable proxy [18] of the rank $r(\mathcal{H}(s^k))$. For tractability, we optimize the pseudo-likelihood approximation of the log-likelihood term in the joint loss. We choose the value of $\lambda \in \{0.0001, 0.001, 0.01, 0.1, 0.9\}$ with 5-fold cross-validation.

To explore the effect of training energy-based models (EBMs) other than the Potts model in this fashion, we use the same procedure to fit an EBM with an energy function parameterized by a two-layer feed-forward neural network with hidden layer sizes of $(300, 100)$ and RELU activations. As with the Potts model, we optimize the pseudo-likelihood approximation of the log-likelihood term in the joint loss.

BLOSUM62 substitution scores

For a mutant sequence, we sum the BLOSUM62 substitution scores between every amino acid in the wild-type sequence and the mutant sequence. The scores are then offset by the a constant such that a score of zero is obtained for the wild-type sequence.

Augmented models

For augmentation, we rely on an already-trained probability density model on a set of evolutionarily-related sequences, that does not get altered. To augment this existing density model, we concatenate the sequence density estimation from the original model together with one-hot encoded site-specific amino acid features, as illustrated in Figure 2.1b. Mathematically, following the same notation for linear models with one-hot encoded features, given a density model with sequence probability distribution $p(s;\varphi)$ and fixed density model parameters φ, the corresponding augmented model is

$$f(s;\varphi,\theta,\beta) = \beta \log p(s;\varphi) + \theta_0 + \sum_{i=1}^{L} \theta_i(s_i),$$

where the parameters $\theta_i \in \mathbb{R}^{|\mathcal{A}|}$, $\theta_0 \in \mathbb{R}$, and $\beta \in \mathbb{R}$ are learned from assay-labelled data. When performing ridge regression with this augmented model, the regularization strength for β was set to make this feature practically unregularized (regularization strength, 10^{-8}), while the other parameters, θ, were regularized using a common strength determined by cross-validation. For density models where computing exact log-likelihoods is challenging, we use one of various approximations. For Potts models and Transformers, we use the pseudo-log-likelihood, and for VAEs, the evidence lower bound (ELBO). Similarly to the setup of a linear model with one-hot encoded site-specific amino acid features, we again used the overparameterized one-hot site-specific amino acid encoding with $|\mathcal{A}|$ features per position (one binary feature for each amino acid possibility per position) in combination with ℓ_2 regularization. See the section on "Effects of ℓ_2 regularization" above in Methods for more details on how this choice influences model prediction on mutations not seen in the training data.

The augmented Potts model as an evolutionary prior for linear regression

One can view the augmented Potts model as a tailoring of the implicit prior in the ℓ_2-regularized (ridge) linear regression with site-specific amino acid features, where the tailoring is based on evolutionary information from the density model. First, note that in the Bayesian, *m*aximum a posteriori (MAP) interpretation of ridge regression, a zero-mean Gaussian prior probability has been given to the ridge regression parameters, $\theta_i(\alpha) \sim \mathcal{N}(0,\tau^2)$. Now consider the augmented Potts model, with already-fitted Potts model, $p(s;\varphi \equiv \{h_i, J_{ij}\})$, with

site-specific parameters, $\{h_i(\alpha)\}$ at position i for amino acid α and analogous coupling parameters, $\{J_{ij}(\alpha_1, \alpha_2)\}$. If in the augmented Potts model, the Potts model density feature was independent from the site-specific amino acid features, one could in a two-step procedure first fit the density-associated parameter, $\beta = \hat{\beta} \in \mathbb{R}$, and having fit that, then fit the remaining parameter vector, θ. Under the independence assumption, the resulting parameter estimates from the two-step procedure would be identical to having estimated them jointly as in regular ridge regression (with no regularization on the density feature, as done for the augmented Potts model). In such a two-step fitting procedure, the second step would correspond to performing ridge regression on the residuals $y' = y - \hat{\beta} \log p(s; \varphi)$. Correspondingly, the implicit prior of ridge regression gets tailored by shifting its mean by an amount, $\hat{\beta}\tilde{h}_i(\alpha)$, corresponding to the Potts model site-specific parameters, to back-interpret the prior over the parameters to be $\theta_i(\alpha) \sim \mathcal{N}(\hat{\beta}\tilde{h}_i(\alpha), \tau^2)$. Note that the effect of the coupling parameters in the Potts model is entirely mediated through the estimate of $\hat{\beta}$. Also note that the assumption of independence of the Potts model density feature from the other features is unlikely to be true in most practical cases; however, this does not necessarily detract from the overall intuitive interpretation.

Model evaluation

For each data set, we randomly sample 20% of the data set as held-out test data. Among the remaining 80% data, we randomly sample $N = 24, 48, 72, 96, \cdots$, or 240 single mutant sequences as training data in separate experiments, or use all single mutant sequences in the 80% training data in the 80-20 split experiments. When computationally feasible (for the linear model, eUniRep regression, and all augmented models), we use five-fold cross-validation to determine hyperparameters. Otherwise, for the fine-tuned Transformer and the integrated Potts model, we set aside 20% of the training data to determine hyperparameters (*i. e.,* the number of fine-tuning epochs for the Transformer and the relative weighting between evolutionary and assay-labelled data for the integrated Potts model). For each fixed sample size, the model evaluation procedure is repeated 20 times with different random seeds for uncertainty estimation. The only exception is for the integrated Potts model, where we only use 5 random seeds due to the long computation time. The 95% confidence intervals of the means are estimated via bootstrapping.

NDCG

Discounted cumulative gain (DCG) and its normalized version NDCG are both widely-used measures of ranking quality in information retrieval. NDCG has also recently been used in the protein engineering community to assess fitness prediction [141]. Intuitively, the NDCG score can be seen as a smoothed version of the mean fitness of the top k predictions, without having to fix an arbitrary k. In the top-k mean metric, the true fitness values of variants are summed with a binary weight of either $1/k$ or 0 depending on whether the variant is in the

top k or not. Instead of the binary weight, NDCG sums over the variants with a smoothed logarithmic weight.

Given a model's prediction on M protein variants, to calculate DCG we first sort all variants by predicted scores into an ordered rank list $\hat{y}_1 \geq \hat{y}_2 \geq \cdots \geq \hat{y}_M$. Then, DCG sums the true fitness values y with a logarithmic discount according to the rank order, where a variant's true fitness value contributes more to the sum if it is more highly ranked,

$$DCG = \sum_{i=1}^{M} \frac{y_i}{log(i+1)}.$$

The best (highest) possible DCG on a set of M protein variants occurs when all protein variants are predicted to be in the exact same rank order as the true fitness. On different data sets, the value of the best possible DCG is different, depending on the true fitness values in the data set. Consequently, the DCG is often normalized to make the values more comparable between data sets. To do so, we first standardize the true scores y to have zero mean and unit variance. Then, we normalize the DCG by dividing by the best possible DCG (*i. e.,* obtained for a perfect ranking) to obtain the NDCG, which therefore lies between 0 and 1 for all data sets.

As an example, consider three protein variants A, B, and C with respectively true fitness values, -0.8, 0.6, 0.2, and predicted fitness values, $0.1, 0.9, -0.1$. First we sort them in predicted order B > A > C. Then the DCG from the predicted model is $0.6/\log(2) + (-0.8)/\log(3) + 0.2/\log(4) = 0.195$, while the ideal DCG from the perfect ranking is $0.6/\log(2) + 0.2/\log(3) + (-0.8)/\log(4) = 0.326$. The ratio between the DCG and the ideal DCG, $0.195/0.326 = 0.598$, is then the NDCG.

Mann–Whitney U test

We use the two-sided nonparametric Mann–Whitney U test for comparing average performance between different methods. The null hypothesis of the U-test is that, for randomly selected values X and Y from two populations, the probability of X being greater than Y is equal to the probability of Y being greater than X. The alternative hypothesis is that one population is stochastically greater than the other. In the context of Figure 2.2, since we are comparing the average performance over all data sets, the population consists of the average performance of a given method computed from different random seeds. This satisfies the assumption that the observations in each population are independent of each other.

FoldX

The FoldX suite [111] evaluates the effect of mutations on the stability of proteins. To derive stability features from FoldX, we use the "total energy" output from the BuildModel command in FoldX 5.0. We used PDB structure 2WUR for the green fluorescent protein

(GFP) following Sarkisyan et al. [110], and PDB structures 6R5K and 2KR4 for the poly(A)-binding protein RRM domain and the ubiquitination factor E4B U-box domain respectively. Unfortunately, among these three proteins, only the GFP has atomic resolution structure determined by X-ray crystallography. The only structures available for the other two domains are determined by nuclear magnetic resonance (NMR) and electron microscopy (EM) at lower resolutions. Higher-resolution structures could potentially make FoldX-derived features more useful on those two domains.

Data availability

All protein fitness data were publicly available through citations. A processed version of these data and our evaluation results are available on Dryad with doi:10.6078/D1K71B. All protein structures used in the study are available publicly with PDB IDs 2WUR, 6R5K, and 2KR4.

Code availability

The code to reproduce the results is available at https://github.com/chloechsu/combining-evolutionary-and-assay-labelled-data.

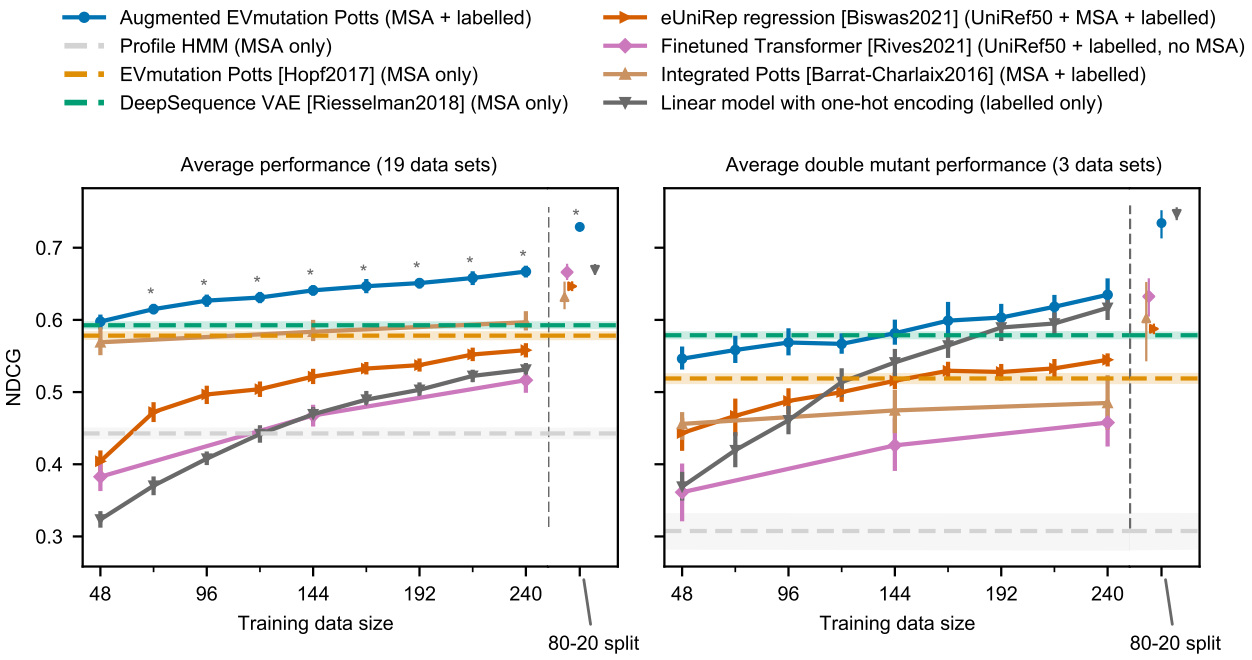

Figure 2.7: Comparison of existing machine learning approaches and the augmented Potts model (NDCG version of Figure 2.2). The left column indicates performance on the full test data, while the right column indicate performance among all double mutants in the test data. Error bars are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits. Asterisks (*) indicate statistical significance ($p < 0.01$) from a two-sided Mann-Whitney U test that the average NDCG from augmented Potts model is different from every other method. The p-values are $p = 3.9\text{e}{-2}, 6.9\text{e}{-7}, 2.2\text{e}{-7}, 7.9\text{e}{-8}, 7.7\text{e}{-4}$, $6.8\text{e}{-8}$, $6.8\text{e}{-8}$, $6.8\text{e}{-8}$ and $7.7\text{e}{-4}$ for training data size $48, 72, 96, \cdots, 240$ and $p = 7.7\text{e}{-4}$ for the 80-20 split.

2.5 Extended Data Figures

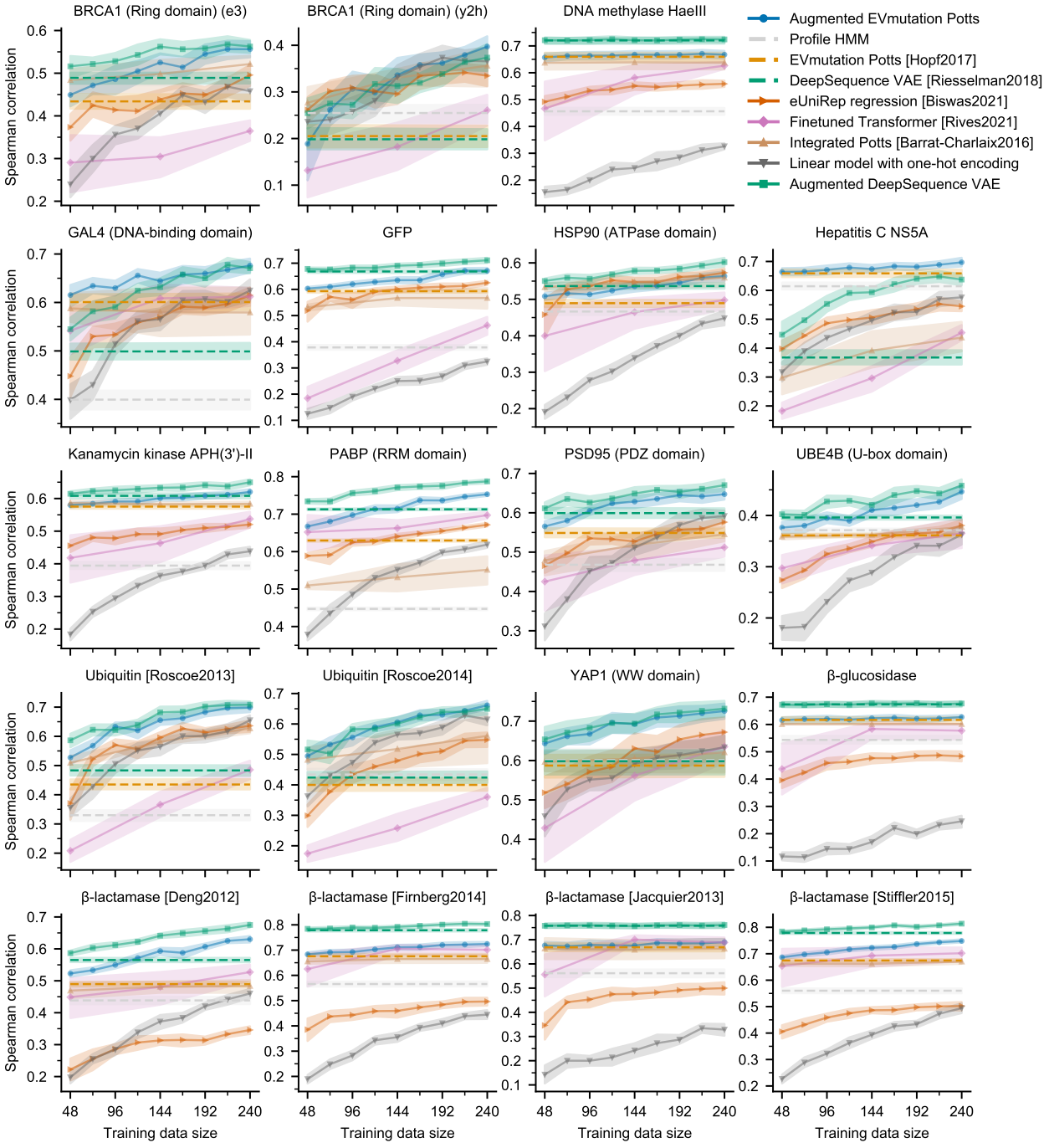

Figure 2.8: Performance on individual data sets when trained on limited labelled data (breakdown of Figure 2.2a). See Figure A.1 for the same breakdown in normalized discounted cumulative gains (NDCG). Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

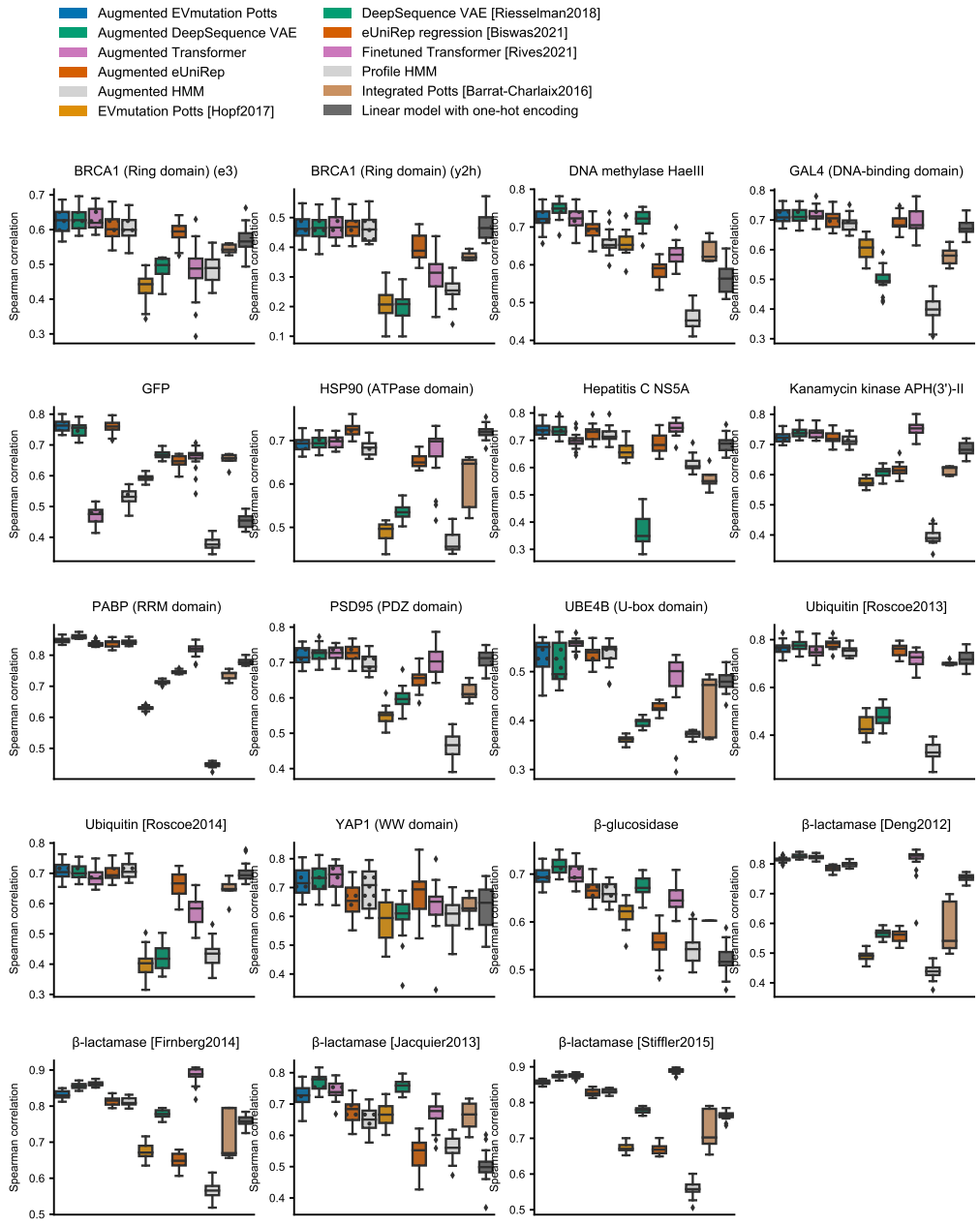

Figure 2.9: Performance on individual data sets when trained on 80% data (breakdown of Figure 2.2a mini-panel). See also Figure A.2 for performance measured by NDCG. Error bars indicate bootstrapped 95% confidence interval from 20 random data splits. Box-and-whisker plots show the first and third quartiles as well as median values. The upper and lower whiskers extend from the hinge to the largest or smallest value no further than 1.5× interquartile range from the hinge.

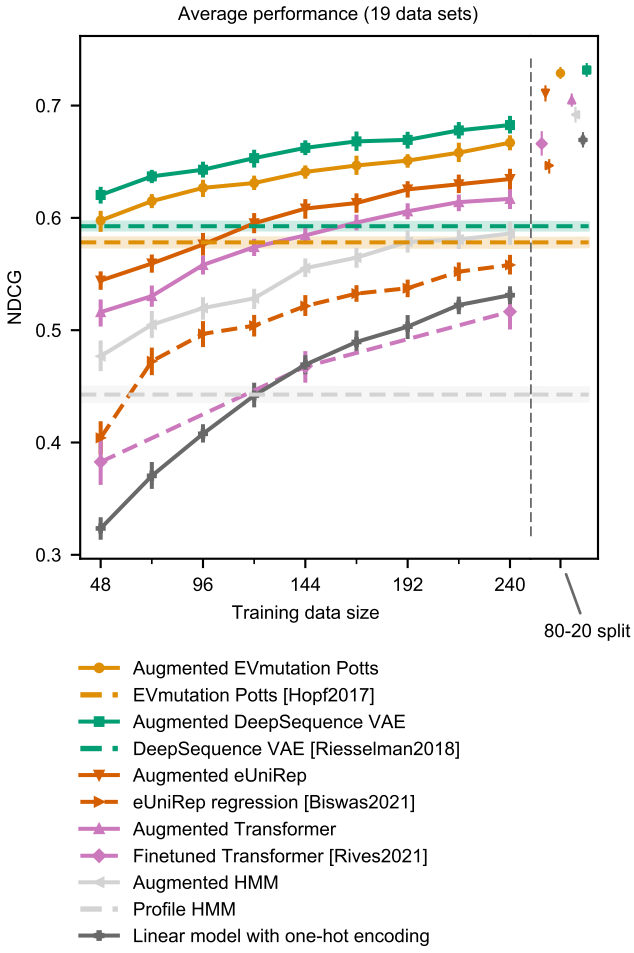

Figure 2.10: Comparison of augmented density models by NDCG (the NDCG version of Figure 2.3a). Dashed lines show evolutionary density model performance, while solid lines in the matching color show the corresponding augmented model. Error bars are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

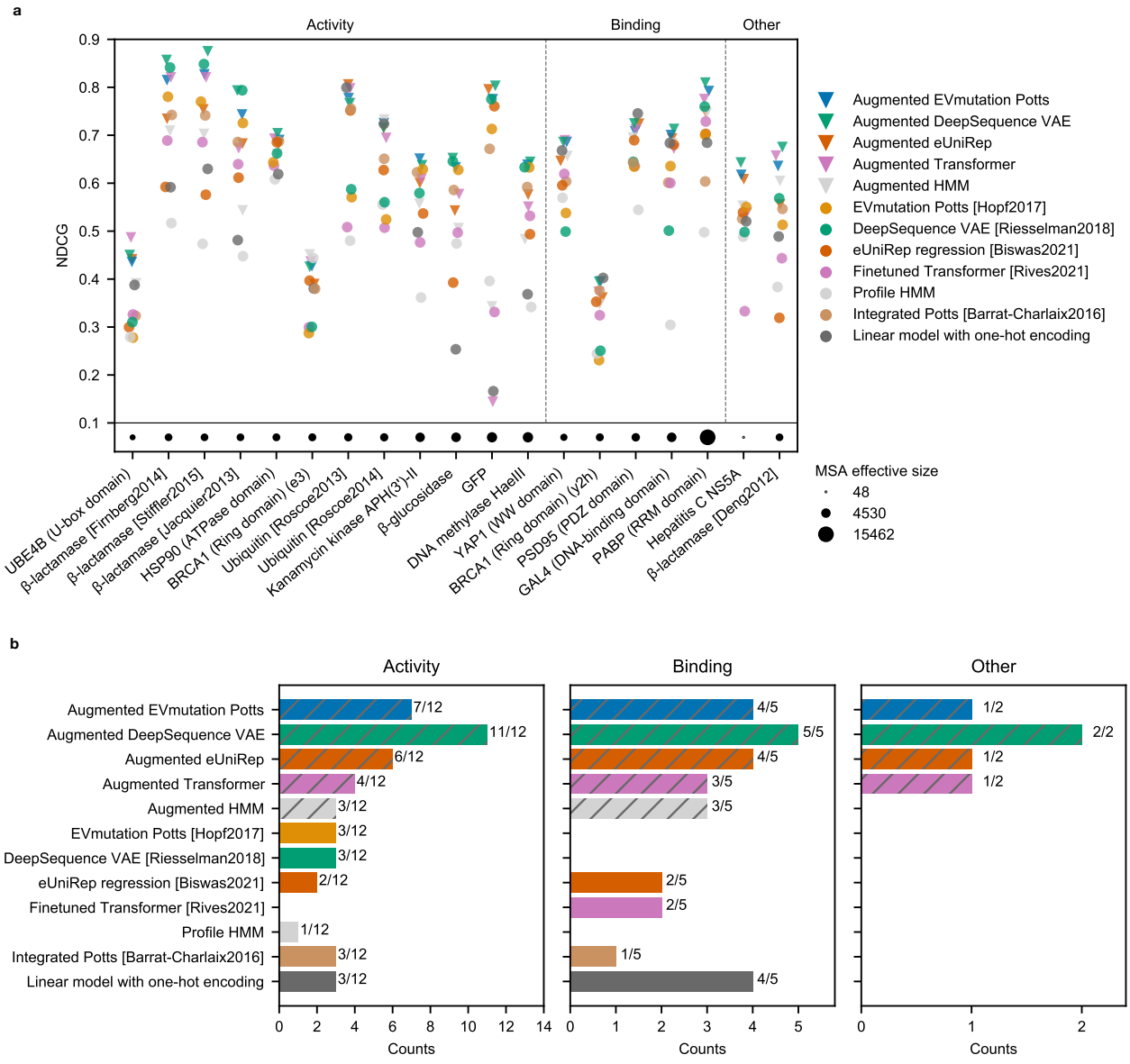

Figure 2.11: Detailed performance analysis by NDCG (the NDCG version of Figure 2.4). (a) Each dot shows the average performance on each data set 20 random training data samples of 240 training examples. The bottom row indicates the effective MSA size after accounting for sequence similarity with sample reweighting at 80% identity cutoff. (b) Best model(s) on each data set. The horizontal axis shows the number of times a given modelling strategy had better NDCG than all other methods. This count is computed by first identifying the top-performing method for any given scenario, and then also counting any other methods that come within the 95% confidence interval of this top performer.

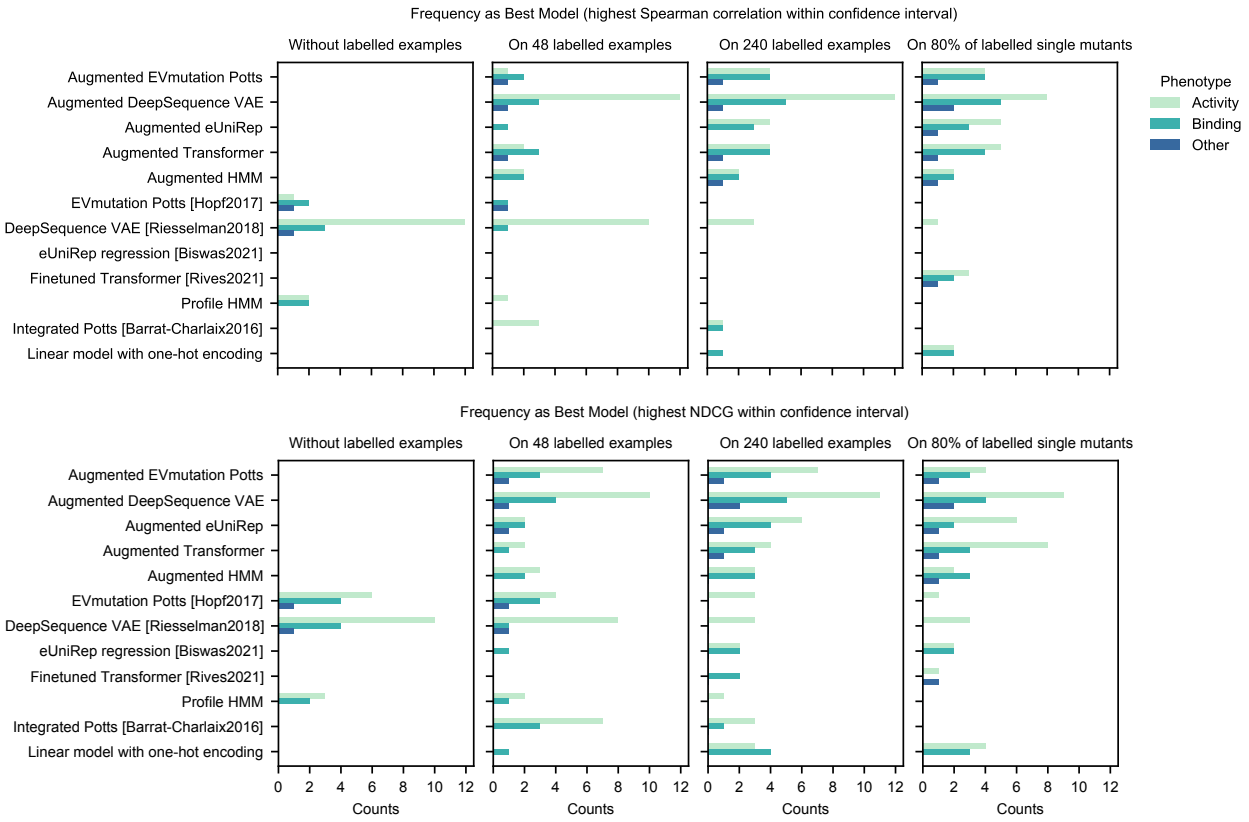

Figure 2.12: Best model(s) on each data set (detailed version of Figure 2.4b with varying training data sizes). (a) Frequency of each method achieving the highest Spearman correlation per data set. The horizontal axis shows the number of times a given modelling strategy had better Spearman correlation than all other methods. This count is computed by first identifying the top-performing method for any given scenario, and then also including any other methods that come within the 95% confidence interval of this top performer. Then both the top performer and those within the 95% confidence interval are counted. Four settings are used: with no labelled data, when training on 48 or 240 labelled single-mutant examples, and in the 80-20 train-test split setting. (b) Frequency of each method achieving the highest NDCG per data set. Similar to (a) but identifying the best model by NDCG instead of Spearman correlation.

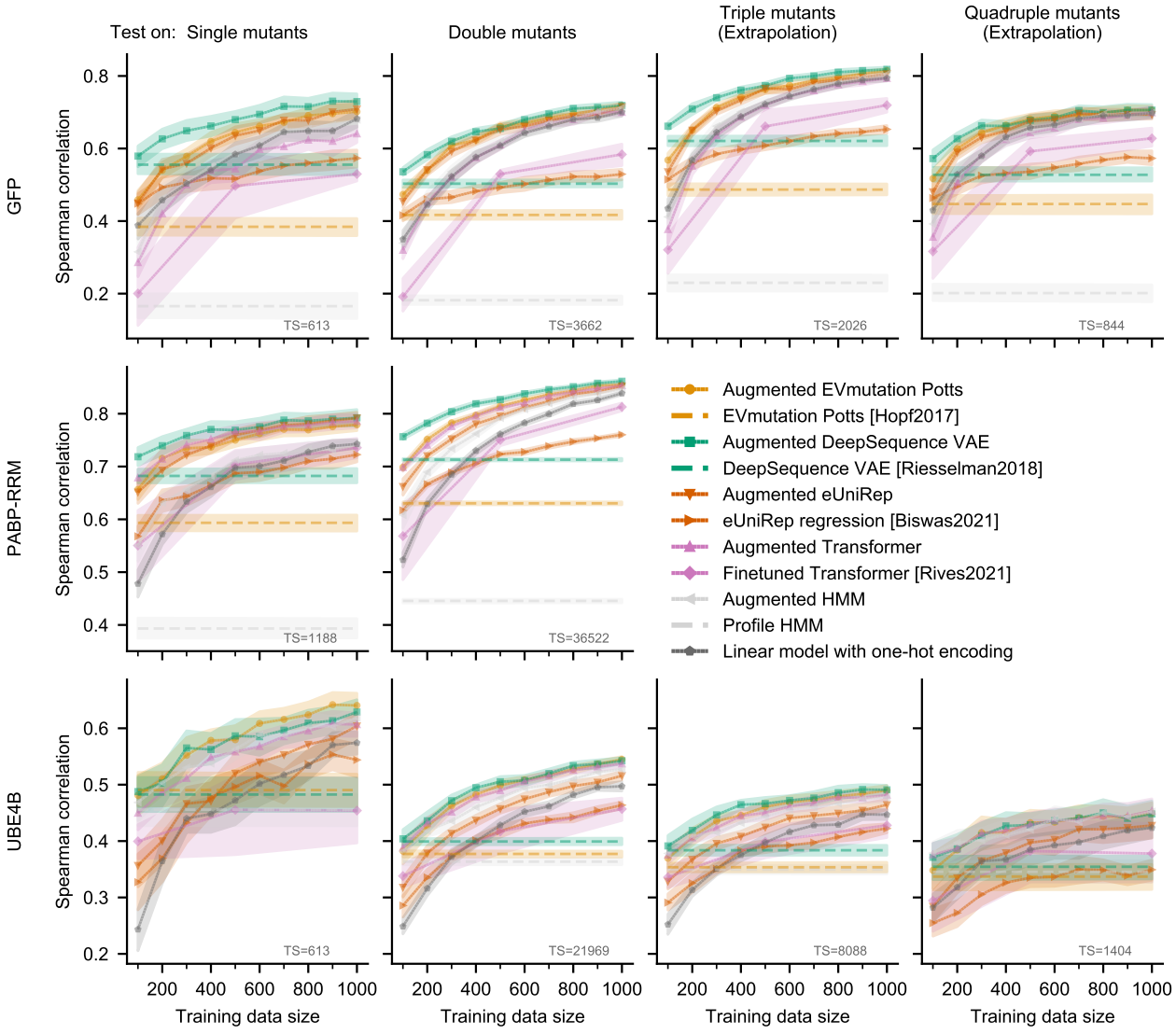

Figure 2.13: Extrapolation performance from single and double mutants to higher-order mutants. Instead of training on single mutants as done in Figure 2.5, here we train on a random sample from both single and double mutants. Each column measures the model performance among single, double, triple, and quadruple mutants separately. See Figure A.6 for NDCG version. The "TS" value indicates the total number of mutants of a particular order in all of the data. For example, "TS=613" for single mutants means there were 613 total single mutants in the data set that we sampled from. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

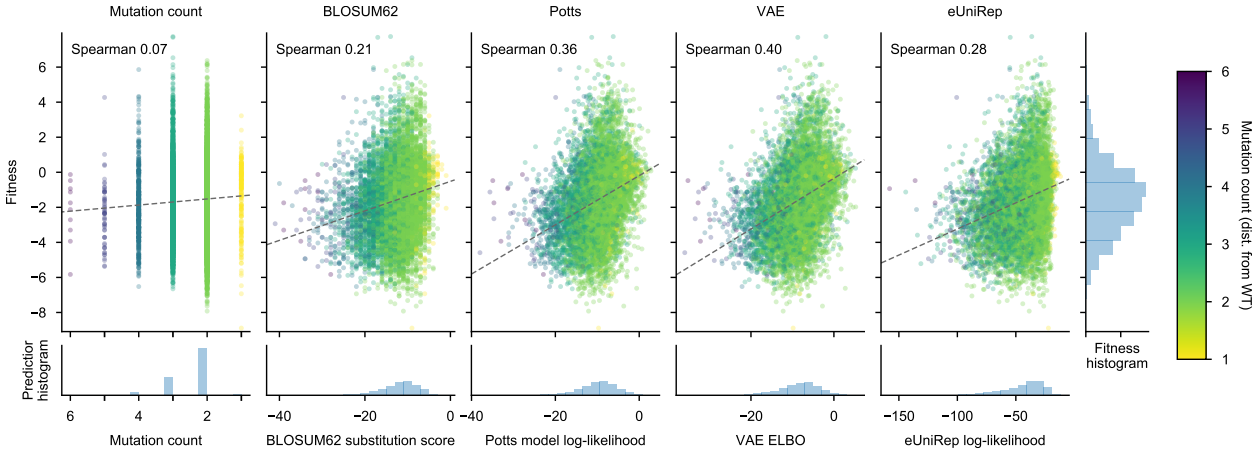

Figure 2.14: Correlations between sequence density estimations of the UBE4B U-box domain and experimentally measured fitness values. Each dot represents a mutant sequence, with darker color indicating further distance from the wild-type. Unlike the case for GFP (Figure 2.6), here mutation count is not a good predictor for fitness.

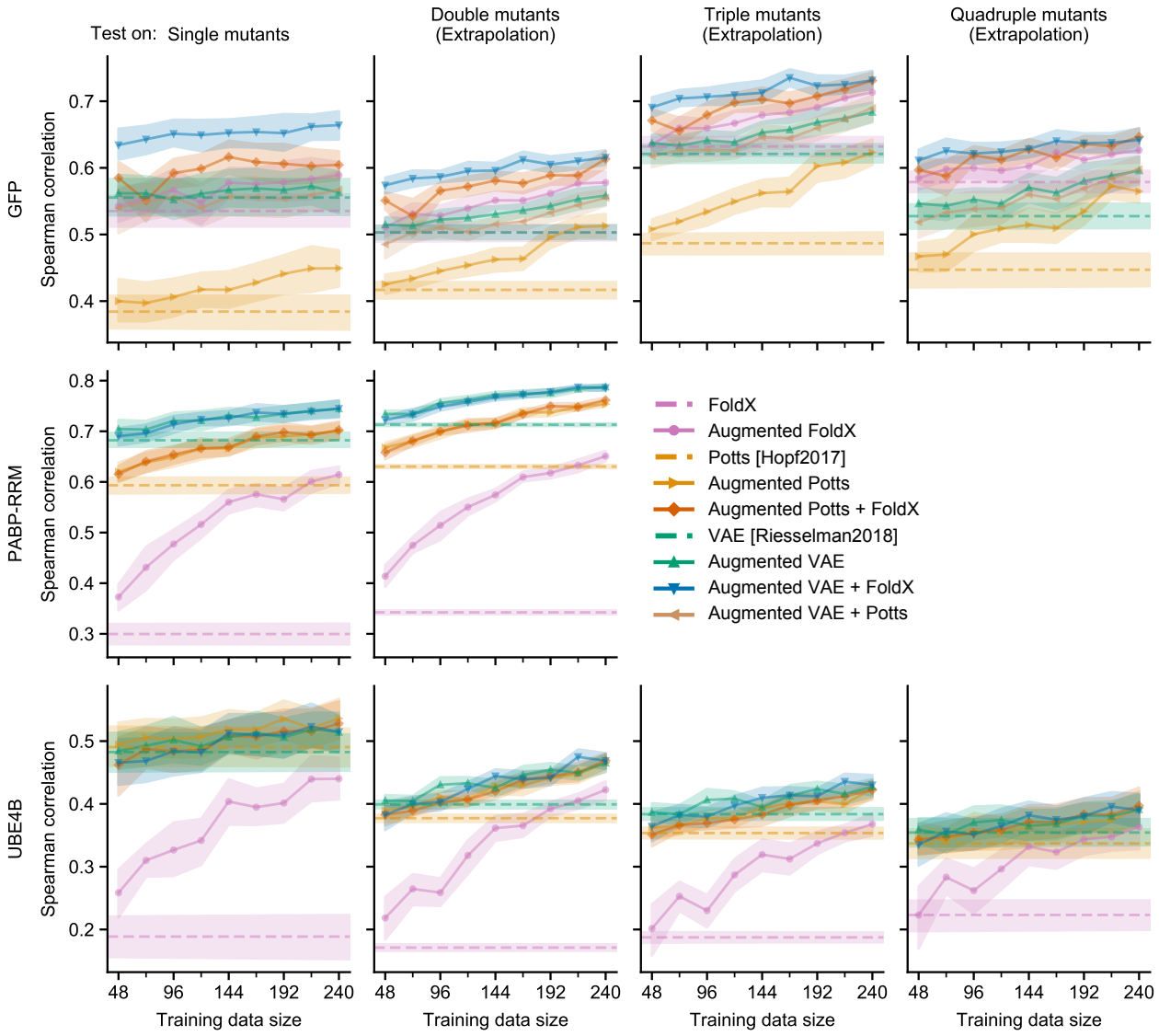

Figure 2.15: FoldX predictions as additional features in augmented models. The FoldX suite evaluates the effect of mutations on the stability of proteins. We evaluate augmented models with FoldX-derived stability features on the three data sets with higher-order mutants. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits. On green fluorescent protein (GFP), stability predictions from FoldX as additional features further improves the performance of augmented models. DeepSequence VAE and FoldX add to each other in augmented models, likely because the two methods incorporate different information. The augmented model with both DeepSequence VAE and FoldX outperform all other augmented models on GFP. On the poly(A)-binding protein RRM domain and the ubiquitination factor E4B U-box domain, however, FoldX-derived features do not add to the augmented VAE model, possibly due to the limited resolution of the available structures for these two domains. See Figure A.7 for comparison by NDCG.

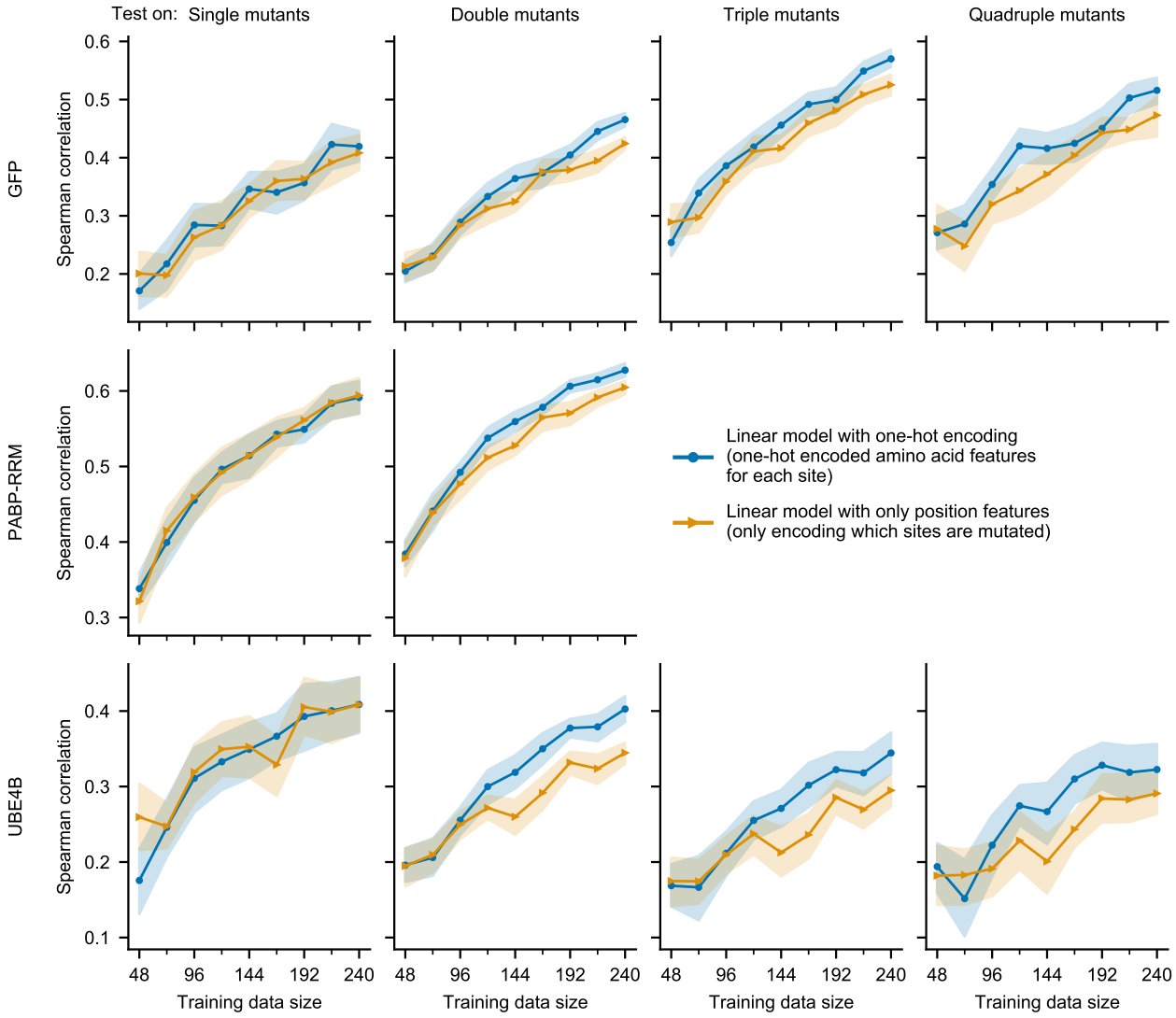

Figure 2.16: Linear model with only position features. Similar to the setup in Figure 2.5, here we train on single mutants and test on higher-order mutants on three case studies: the green fluorescent protein (GFP), the RRM domain of Poly(A)-binding protein (PABP), and the U-box domain of ubiquitination factor E4B (UBE4B). Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits. The linear model with only position-specific features matches the performance of the linear model with one-hot encoded amino acid features on single mutants, but with a slight performance gap on double, triple, and quadruple mutants. This is as expected since all single mutants in the test set have non-overlapping mutations with the train set, while double mutants might have mutations that also appeared in the train set.

Chapter 3

Learning Inverse Folding from Millions of Predicted Structures

Designing novel amino acid sequences that encode proteins with desired properties, known as *de novo protein design*, is a central challenge in bioengineering [59]. The most well-established approaches to this problem use an energy function which directly models the physical basis of a protein's folded state [3].

Recently a new class of deep learning based approaches has been proposed, using generative models to predict sequences for structures [61, 126, 7, 67], generate backbone structures [6, 36], jointly generate structures and sequences [9, 138], or model sequences directly [105, 84, 117, 45, 21, 27]. The potential to learn the rules of protein design directly from data makes deep generative models a promising alternative to current physics-based energy functions.

However, the relatively small number of experimentally determined protein structures places a limit on deep learning approaches. Experimentally determined structures cover less than 0.1% of the known space of protein sequences. While the UniRef sequence database [128] has over 50 million clusters at 50% sequence identity; as of January 2022, the Protein Data Bank (PDB) [14] contains structures for fewer than 53,000 unique sequences clustered at the same level of identity.

Here in this chapter we explore whether predicted structures can be used to overcome the limitation of experimental data. With progress in protein structure prediction [71, 39, 11], it is now possible to consider learning from predicted structures at scale. Predicting structures for the sequences in large databases can expand the structural coverage of protein sequences by orders of magnitude. To train an inverse model for protein design, we predict structures for 12 million sequences in UniRef50 using AlphaFold2.

This chapter focuses on the problem of predicting sequences from backbone structures, known as *inverse folding* or fixed backbone design. We approach inverse folding as a sequence-to-sequence problem [61], using an autoregressive encoder-decoder architecture, where the model is tasked with recovering the native sequence of a protein from the coordinates of its backbone atoms.

We make use of the large number of sequences with unknown structures by adding them as additional training data, conditioning the model on predicted structures when the experimental structures are unknown (Figure 3.1). This approach parallels back-translation [113, 34] in machine translation, where predicted translations in one direction are used to improve a model in the opposite direction. Back-translation has been found to effectively learn from extra target data (i.e. sequences) even when the predicted inputs (i.e. structures) are of low quality.

We find that existing approaches have been limited by data. While current state-of-the-art inverse folding models degrade when training is augmented with predicted structures, much larger models and different model architectures can effectively learn from the additional data, leading to an improvement of nearly 10 percentage points in the recovery of sequences for structurally held out native backbones.

We evaluate models on fixed backbone design benchmarks from prior work, and assess the generalization capabilities across a series of tasks including design of complexes and binding sites, partially masked backbones, and multiple conformations. We further consider the use of the models for zero-shot prediction of mutational effects on protein function and stability, complex stability, and binding affinity.

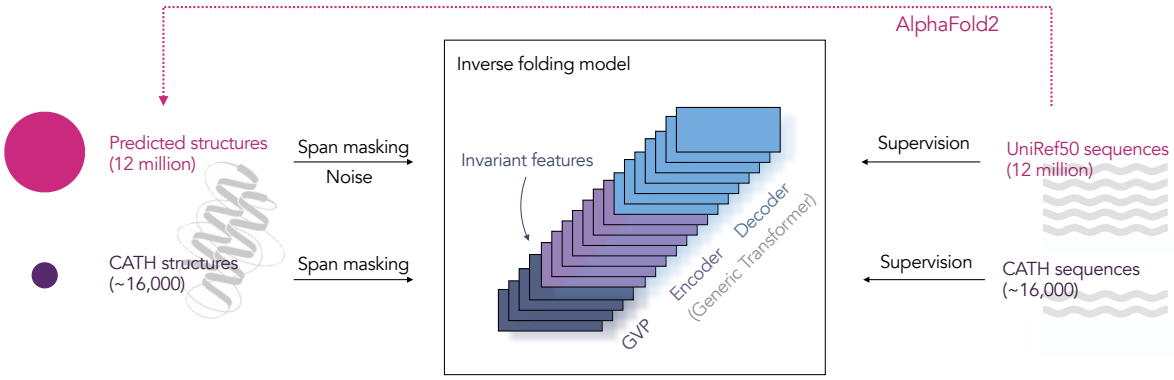

Figure 3.1: Augmenting inverse folding with predicted structures. To evaluate the potential for training protein design models with predicted structures, we predict structures for 12 million UniRef50 protein sequences using AlphaFold2 [71]. An autoregressive inverse folding model is trained to perform fixed-backbone protein sequence design. Train and test sets are partitioned at the topology level, so that the model is evaluated on structurally held-out backbones. We compare transformer models having invariant geometric input processing layers, with fully geometric models used in prior work. Span masking and noise is applied to the input coordinates.

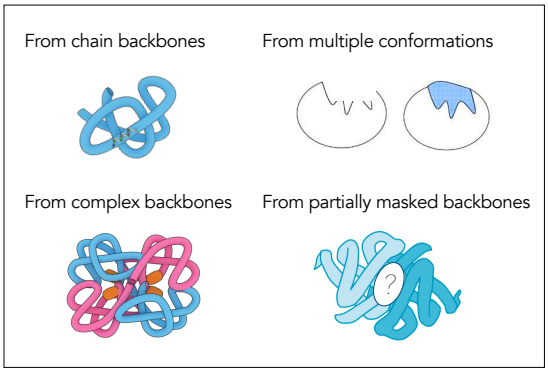

Figure 3.2: Illustration of the protein design tasks considered.

3.1 Learning Inverse Folding from Predicted Structures

The goal of inverse folding is to design sequences that fold to a desired structure. In this work, we focus on the backbone structure without considering side chains. While each of the 20 amino acid has a specific side chain, they share a common set of atoms that make up the amino acid backbone. Among the backbone atoms, we choose the N, Cα (alpha Carbon), and C atom coordinates to represent the backbone.

Using the structures of naturally existing proteins we can train a model for this task by supervising it to predict the protein's native sequence from the coordinates of its backbone atoms in three-dimensional space. Formally we represent this problem as one of learning the conditional distribution $p(Y|X)$, where for a protein of length n, given a sequence X of spatial coordinates $(x_1, \ldots, x_i, \ldots, x_{3n})$ for each of the backbone atoms N, $C\alpha$, C in the structure, the objective is to predict Y the native sequence $(y_1, \ldots, y_i, \ldots, y_n)$ of amino acids. This density is modeled autoregressively through a sequence-to-sequence encoder-decoder:

$$p(Y|X) = \prod_{i=1}^{n} p(y_i|y_{i-1}, \ldots, y_1; X) \tag{3.1}$$

We train a model by minimizing the negative log likelihood of the data. We can design sequences by sampling, or by finding sequences that maximize the conditional probability given the desired structure.

Data

Predicted structures We generate 12 million structures for sequences in UniRef50 to explore how predicted structures can improve inverse folding models. To select sequences for structure prediction we first use MSA Transformer [102] to predict distograms for MSAs of

all UniRef50 sequences. We rank the sequences by distogram LDDT scores [112] as a proxy for the quality of the predictions. We take the top 12 million sequences not longer than five hundred amino acids and forward fold them using the AlphaFold2 model with a final Amber [54] relaxation. This results in a predicted dataset approximately 750 times the size of the training set of experimental structures (Appendix B.1).

Training and evaluation data We evaluate models on a structurally held-out subset of CATH [94]. We partition CATH at the topology level with an 80/10/10 split resulting in 16153 structures assigned to the training set, 1457 to the validation set, and 1797 to the test set. Particular care is required to prevent leakage of information in the test set via the predicted structures. We use Gene3D topology classification [79] to filter both the sequences used for supervision in training, as well as the MSAs used as inputs for AlphaFold2 predictions (Appendix B.1). We also perform evaluations on a smaller subset of the CATH test set that has been additionally filtered by TM-score using Foldseek [74] to exclude any structures with similarity to those in the training set (Appendix B.4).

Model architectures

We study model architectures using Geometric Vector Perceptron (GVP) layers [67] that learn rotation-equivariant transformations of vector features and rotation-invariant transformations of scalar features.

We present results for three model architectures: (1) GVP-GNN from Jing et al. [67] which is currently state-of-the-art on inverse folding; (2) a GVP-GNN with increased width and depth (GVP-GNN-large); and (3) a hybrid model consisting of a GVP-GNN structural encoder followed by a generic transformer (GVP-Transformer). All models used in evaluations are trained to convergence, with detailed hyperparameters listed in Table B.1.

In inverse folding, the predicted sequence should be independent of the reference frame of the structural coordinates. For any rotation and translation T of the input coordinates, we would like for the model's output to be invariant under these transformations, i.e., $p(Y|X) = p(Y|TX)$. Both the GVP-GNN and GVP-Transformer inverse folding models studied in this work are invariant (Appendix B.3).

GVP-GNN We start with the GVP-GNN architecture with 3 encoder layers and 3 decoder layers as described in [67], with the vector gates described in [66] (GVP-GNN, 1M parameters). As inputs to GVP-GNN, protein structures are represented as proximity graphs where each amino acid corresponds to a node in the graph. The node features are a combination of scalar node features derived from dihedral angles and vector node features derived from the relative positions of the backbone atoms, while the edge features capture the relative positions of nearby amino acids.

When trained on predicted structures, we find a deeper and wider version of GVP-GNN with 8 encoder layers and 8 decoder layers (GVP-GNN-large, 21M parameters) performs better. Scaling GVP-GNN further did not improve model performance in preliminary experiments (Figure 3.6c).

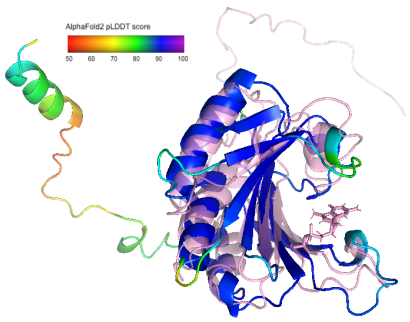

Figure 3.3: Example AlphaFold prediction compared with experimental structure for a UniRef50 sequence (UniRef50: P07260; PDB: 1AP8). The experimental structure is shown as pink with transparency. The prediction is coloured by the pLDDT confidence score, with blue in high-confidence regions.

GVP-Transformer We use GVP-GNN encoder layers to extract geometric features, followed by a generic autoregressive encoder-decoder Transformer [134]. In GVP-GNN, the input features are translation-invariant and each layer is rotation-equivariant. We perform a change of basis on the vector features from GVP-GNN into local reference frames defined for each amino acid to derive rotation-invariant features (Appendix B.3). In ablation studies increasing the number of GVP-GNN encoder layers improves the overall model performance (Figure B.3), indicating that the geometric reasoning capability in GVP-GNN is complementary to the Transformer layers. Scaling improves performance up to a 142M-parameter GVP-Transformer model with 4 GVP-GNN encoder layers, 8 generic Transformer encoder layers, and 8 generic Transformer decoder layers (Figure 3.6c).

Training

Combining experimental and predicted data During training, in each epoch we mix the training set of experimentally derived structures (∼16K structures) with a 10% random sample of the AlphaFold2-predicted training set (10% of 12M), resulting in a 1:80 experimental:predicted data ratio. For larger models, a high ratio of predicted data during training helps prevent overfitting on the smaller experimental train set (Figure 3.6b). While adding predicted data improves performance, training only on predicted data leads to substantially worse performance (Table B.4).

The loss is equally weighted for each amino acid in target sequences. We mask out predicted input coordinates with AlphaFold2 confidence score (pLDDT) below 90, around 25% of the predicted coordinates. See Figure 3.3 for visualization of the pLDDT confidence score. Most often these low confidence regions are at the start and the end of sequences and may correspond to disordered regions. We prepend one token at the beginning of each

		Perplexity			Recovery %		
Model	Data	Short	Single-chain	All	Short	Single-chain	All
Natural frequencies		18.12	18.03	17.97	9.6%	9.0%	9.5%
Structured GNN	CATH	7.91	6.48	6.49	31.5%	37.1%	37.1%
GVP-GNN	CATH	7.14	5.36	5.43	34.0%	42.7%	42.2%
	+ AlphaFold2	8.55	6.17	6.06	29.5%	38.2%	38.6%
GVP-GNN-large	CATH	7.68	6.12	6.17	32.6%	39.4%	39.2%
	+ AlphaFold2	6.11	4.09	4.08	**38.3%**	50.8%	50.8%
GVP-Transformer	CATH	8.18	6.33	6.44	31.3%	38.5%	38.3%
	+ AlphaFold2	**6.05**	**4.00**	**4.01**	38.1%	**51.5%**	**51.6%**

Table 3.1: Fixed backbone sequence design. Evaluation on the CATH 4.3 topology split test set. Models are compared on the basis of per-residue perplexity (lower is better; lowest perplexity bolded) and sequence recovery (higher is better; highest sequence recovery bolded). Large models can make better use of the predicted UniRef50 structures. The best model trained with predicted structures (GVP-Transformer) improves sequence recovery by 8.9 percentage points over the best model (GVP-GNN) trained on CATH only.

sequence to indicate whether the structure is experimental or predicted. For each residue we provide the pLDDT confidence score from AlphaFold2 as a feature encoded by Gaussian radial basis functions.

Adding Gaussian noise at the scale of 0.1 angstroms to the predicted structures during training slightly improves performance (Table B.3). This finding is consistent with Edunov et al. [34], who observe that backtranslation with sampled or noisy synthetic data provides a stronger training signal than maximum a posteriori (MAP) predictions.

Span masking To enable sequence design for partially masked backbones, we introduce backbone masking during training. We experiment with both independent random masking and span masking. In natural language processing, span masking improves performance over random masking [70]. We randomly select continuous spans of up to 30 amino acids until 15% of input backbone coordinates are masked. The communication patterns in the geometric layers are adapted to account for masking with details in Appendix B.2. Span masking improves the performance of GVP-Transformer both on unmasked backbones (Table B.3) and on masked regions (Figure 3.4).

3.2 Results

We evaluate models across a variety of benchmarks in two overall settings: fixed backbone sequence design and zero-shot prediction of mutation effects. For fixed backbone design, we start with evaluation in the standard setting [61, 67] of sequence design given all backbone coordinates. Then, we make the sequence design task more challenging along three dimensions: (1) introducing masking on coordinates; (2) generalization to protein complexes; and (3) conditioning on multiple conformations. Additionally, we show that inverse folding

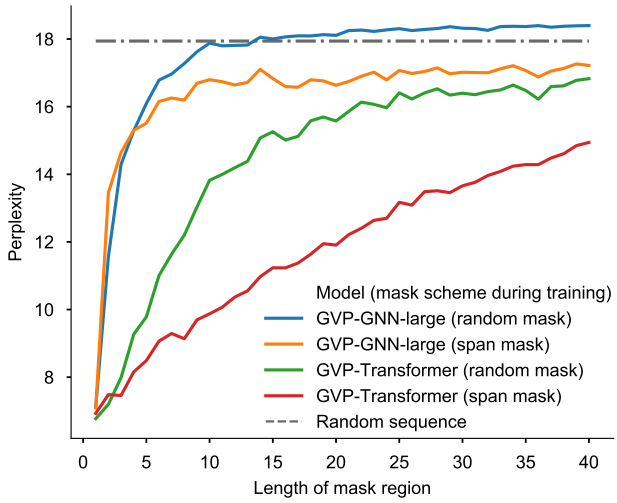

Figure 3.4: Perplexity on regions of masked coordinates of different lengths. The GVP-GNN architecture degrades to the perplexity of the background distribution for masked regions of more than a few tokens, while GVP-Transformer maintains moderate accuracy on long masked spans, especially when trained on masked spans.

models are effective zero-shot predictors for protein complex stability, binding affinity, and insertion effects.

Fixed backbone protein design

We begin with the task of predicting the native protein sequence given its backbone atom (N, Cα, C) coordinates. Perplexity and sequence recovery on held-out native sequences are two commonly used metrics for this task. Perplexity measures the inverse likelihood of native sequences in the predicted sequence distribution (low perplexity for high likelihood). Sequence recovery (accuracy) measures how often sampled sequences match the native sequence at each position. To maximize sequence recovery, the predicted sequences are sampled with low temperature $T = 1e{-}6$ from the model. While the model is calibrated (Figure B.7), a lower temperature results in sequences with higher likelihoods (and hence typically higher sequence recovery) and lower diversity. Empirically at temperature as low as $1e{-}6$ the sampling is almost deterministic. Table 3.1 compares models using the perplexity and sequence recovery metrics on the structurally held-out backbones.

We observe that current state-of-the-art inverse folding models are limited by the CATH training set. Scaling the current 1M parameter model (GVP-GNN) to 21M parameters (GVP-GNN-large) on the CATH dataset results in overfitting with a degradation of sequence recovery from 42.2% to 39.2% (Table 3.1). On the other hand, the current model at the 1M parameter scale cannot make use of the predicted structures: training GVP-GNN with predicted structures results in a degradation to 38.6% sequence recovery (Table 3.1), with

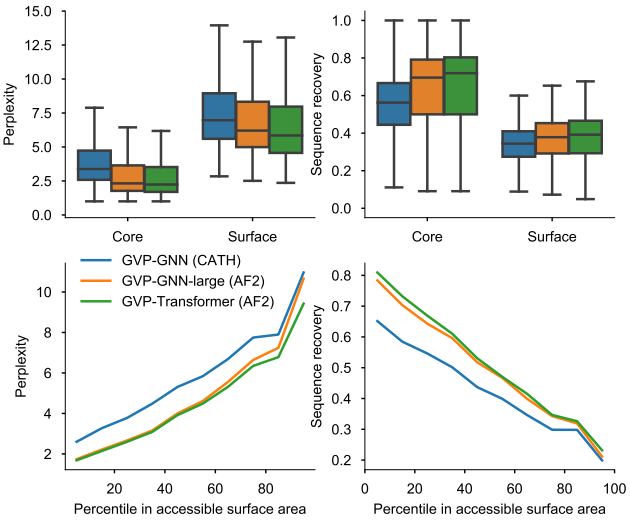

Figure 3.5: Comparison of perplexity and sequence recovery by structural context according to two different measures: number of neighbors (top) and solvent accessible surface area (bottom). Top: Breakdown for core and surface residues. Residues are categorized by density of neighboring Cα atoms within 10A of the central residue Cα atom (core: $\geq$ 24 neighbors; surface: $<$ 16 neighbors). Each box shows the distribution of perplexities for the core or surface residues across different sequences. Bottom: Perplexity and sequence recovery as a function of solvent accessible surface area. Increased sequence recovery for buried residues suggests the model learns dense hydrophobic packing constraints in the core.

performance worsening with increasing numbers of predicted structures in training (Figure 3.6a).

Larger models benefit from training on the AlphaFold2-predicted UniRef50 structures. Training with predicted structures increases sequence recovery from 39.2% to 50.8% for GVP-GNN-large and from 38.3% to 51.6% for GVP-Transformer over training only on the experimentally derived structures. The improvements are also reflected in perplexity. Similar improvements are observed on the test subset filtered by TM-score (Table B.2). The best model trained with UniRef50 predicted stuctures, GVP-Transformer, improves sequence recovery by 9.4 percentage points over the best model, GVP-GNN, trained on CATH alone.

As there are many sequences that can fold to approximately the same structure, even an ideal protein design model will not have 100% native sequence recovery. We observe that the GVP-GNN-large and GVP-Transformer models are well-calibrated (Figure B.7). The substitution matrix between native sequences and model-designed sequences resembles the BLOSUM62 substitution matrix (Figure B.6), albeit noticeably sparser for the amino acid Proline.

When we break down performance on core residues and surface residues, as expected, core residues are more constrained and have a high native sequence recovery rate of 72%, while surface residues are not as constrained and have a lower sequence recovery of 39%

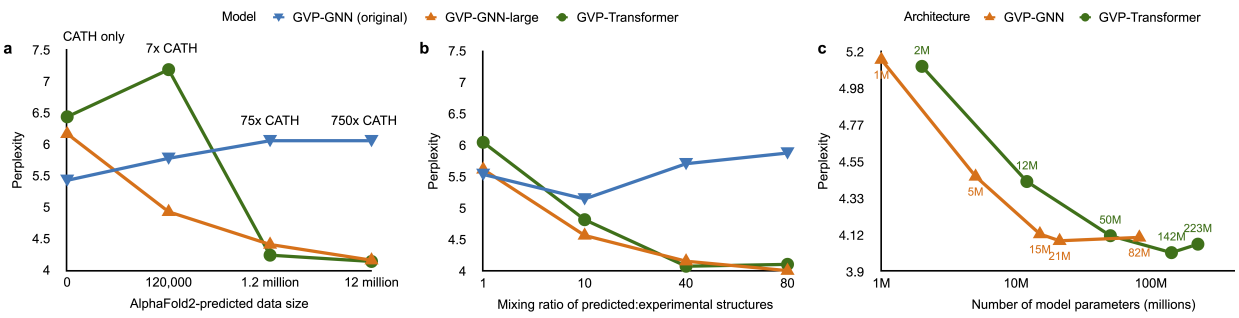

Figure 3.6: Ablation studies on training data. (a) Effect of increasing the number of predicted structures. The original GVP-GNN degrades with training on additional data, but GVP-GNN-large and GVP-Transformer improve with increasing numbers of predicted structures. (b) Effect of increasing the mixing ratio during training between predicted and experimental structures. A higher ratio of predicted structures improves performance for both GVP-GNN-large and GVP-Transformer. (c) GVP-GNN and GVP-Transformer model size.

(Figure 3.5; top). Generally perplexity increases with the solvent accessible surface area (Figure 3.5; bottom). Despite the lower sequence recovery on the surface, sampled sequences do tend not to have hydrophobic residues on the surface (Figure B.8).

As an example of inverse folding of a structurally-remote protein, we re-design the receptor binding domain (RBD) sequence of the SARS-CoV-2 spike protein (PDB: 6XRA and 6VXX; illustrated in Figure B.5) with the two models. The SARS-CoV-2 spike protein has no match to the training data with TM-score above 0.5. Both GVP-GNN and GVP-Transformer achieve high sequence recovery (49.7% and 53.6%) for the native RBD sequence (Table B.6).

While perplexity and sequence recovery are informative metrics, low perplexity and high sequence recovery do not necessarily guarantee structural similarity. One empirically observed failure mode in sampled sequences is repetition of the same amino acid, e.g. EEEEEEE. It would be interesting to further identify more failure modes by studying the experimental or predicted structures of sampled sequences.

Partially-masked backbones We evaluate the models on partial backbones. While masking during training does not significantly change test performance on unmasked backbones (Table B.3), masking does enable models to non-trivially predict sequences for mask regions. Although GVP-GNN-large has low perplexity on short-length masks, its performance quickly degrades to the perplexity of the background distribution on masks longer than 5 amino acids (Figure 3.4). By contrast, the GVP-Transformer model maintains moderate performance even on longer masked regions, with less degradation if trained with span masking instead of independent random masking (Figure 3.4).

Model	Perplexity	
	Chain	Complex
Natural frequencies	17.93	
GVP-GNN	7.80	5.37
GVP-GNN-large+AF2	**6.32**	3.90
GVP-Transformer+AF2	**6.32**	**3.81**

Table 3.2: Sequence design performance on complexes in the CATH topology test split when given the backbone coordinates of only a chain ("Chain" column) and when given all backbone coordinates of the complex ("Complex" column). The perplexity is evaluated on the same chain in the complex for both columns.

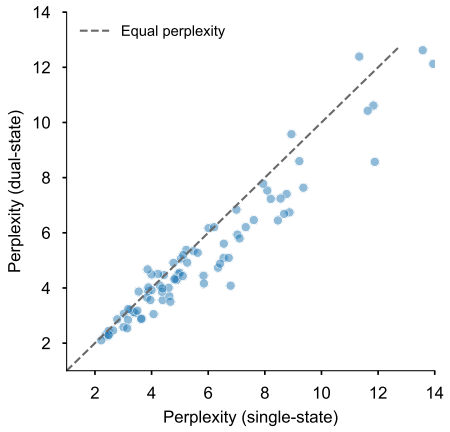

Figure 3.7: Dual-state design. GVP-Transformer conditioned on two conformations results in lower sequence perplexity at locally flexible residues than single-conformation conditioning for structurally held-out proteins in PDBFlex (see Appendix B.5 for details).

Protein complexes Although the training data only consists of single chains, we find that models generalize to multi-chain protein complexes. We represent complexes by concatenating the chains together with 10 mask tokens between chains, and place the target chain for sequence design at the beginning during concatenation. We include all complexes in the CATH 4.3 test set up to 1000 amino acids in length. For chains that are part of a protein complex, there is a substantial improvement in perplexity of both models when given the full complex coordinates as input, versus only the single chain (Table 3.2 and Figure B.4), suggesting that both GVP-GNN and GVP-Transformer can make use of inter-chain information from amino acids that are close in 3D structure but far apart in sequence.

	Spearman correlation			
Model	No coords	No RBM coords	No ACE2 coords	All coords
ESM-1v	0.03			
ESM-1b	0.02			
ESM-MSA-1b (few-shot)	**0.51**			
GVP-GNN		-0.10	0.50	0.60
GVP-GNN-large+AF2		-0.05	0.52	**0.69**
GVP-Transformer+AF2		-0.06	**0.53**	0.64

Table 3.3: Zero-shot performance on binding affinity prediction for the receptor binding domain (RBD) of SARS-CoV-2 Spike, evaluated on ACE2-RBD mutational scan data [122]. The zero-shot predictions are based on the sequence log-likelihood for the receptor binding motif (RBM), which is the portion of the RBD in direct contact with ACE2 [76]. We evaluate in four settings: 1) Given sequence data alone ("No coords"); 2) Given backbone coordinates for both ACE2 and the RBD but excluding the RBM and without sequence ("No RBM coords"); 3) Given the full backbone for the RBD but no information for ACE2 ("No ACE2 coords"); and 4) Given all coordinates for the RBD and ACE2.

Multiple conformations Multi-state design is of interest for engineering enzymes and biosensors [77, 100]. Some proteins exist in multiple distinct folded forms in equilibrium, while other proteins may exhibit distinct conformations when binding to partner molecules.

For a backbone X, the inverse folding model predicts a conditional distribution $p(Y|X)$ over possible sequences Y for the backbone. To design a protein sequence compatible with two states A and B, we would like find sequences with high likelihoods in the conditional distributions $p(Y|A)$ and $p(Y|B)$ for each state. We use the geometric average of the two conditional likelihoods as a proxy for the desired distribution $p(Y|A, B)$ conditioned on the sequence being compatible with both states.

We compare single-state and multi-state sequence design performance on 87 test split proteins with multiple conformations in the PDBFlex dataset [56]. On locally flexible residues, multi-state design results in lower sequence perplexity than single-state design (Figure 3.7). See Appendix B.5 for more details on the PDBFlex data.

Zero-shot predictions

We next show that inverse folding models are effective zero-shot predictors of mutational effects across practical design applications, including prediction of complex stability, binding affinity, and insertion effects. To score the effect of a mutation on a particular sequence, we use the ratio between likelihoods of the mutated and wildtype sequences according to the inverse folding model, given the experimentally determined wildtype structure. Exact

likelihood evaluations are possible from both GVP-GNN and GVP-Transformer as they are both based on autoregressive decoders. We then compare these likelihood ratio scores to experimentally-determined fitness values measured on the same set of sequences.

***de novo* mini-proteins** Rocklin et al. [107] performed deep mutational scans across a set of *de novo* designed mini-proteins with 10 different folds measuring the stability in response to point mutations. The likelihoods of inverse folding models have been shown to correlate with experimentally measured stability using this dataset [61, 67]. We evaluate the GVP-Transformer and GVP-GNN-large models on the same mutational scans, and observe improvements in stability predictions from using predicted structures as training data for 8 out of 10 folds in the dataset (Table B.5). Further details are in Appendix B.5.

Complex stability We evaluate models on zero-shot prediction of mutational effects on protein complex interfaces, using the Atom3D benchmark [129] which incorporates binding free energy changes in the SKEMPI database [63] as a binary classification task. We find that sequence log-likelihoods from GVP-GNN are effective zero-shot predictors of stability changes of protein complexes even without predicted structures as training data (Table B.7), performing comparably to the best supervised method which uses transfer learning. While we observe a substantial improvement in perplexity when predicted structures are added to training (Table 3.2), this does not further improve complex stability prediction for the single-point mutations in SKEMPI (Table B.7), indicating potential limitations of evaluating models only on single-point mutations.

Binding affinity While the SKEMPI dataset features one mutation entry per protein, we also want to evaluate whether inverse folding models can rank different mutations on the same protein, potentially enabling binding-affinity optimization, which is an important task in therapeutic design. We assess whether inverse folding models can predict mutational effects on binding by leveraging a dataset generated by Starr et al. [122] in which all single amino acid substitutions to the SARS-CoV-2 receptor binding domain (RBD) were experimentally measured for binding affinity to human ACE2. Given potential applications to interface optimization or design, we focus on mutations within the receptor binding motif (RBM), the portion of the RBD in direct contact with ACE2 [76]. When given all RBD and ACE2 coordinates, the best inverse folding model produces RBD-sequence log-likelihoods that have a Spearman correlation of 0.69 with experimental binding affinity measurements (Table 3.3). We observe weaker correlations when not providing the model with ACE2 coordinates, indicating that inverse folding models take advantage of structural information in the binding partner. When masking RBM coordinates (69 of 195 residues, a longer span than masked during model training), we no longer observe correlation between RBD log-likelihood and binding affinity, indicating that the model relies on structural information at the interface to identify interface designs that preserve binding. Zero-shot prediction via inverse folding outperforms methods for sequence-based variant effect prediction, which use

the likelihood ratio between the mutant and wildtype amino acids at each position to predict the impact of a mutation on binding affinity. These likelihoods are inferred by masked language models, ESM-1b, ESM-1v, and ESM-MSA-1b, as described by Meier et al. [88] (Table 3.3); additional details are given in Appendix B.5.

Sequence insertions Using masked coordinate tokens at insertion regions, inverse folding models can also predict insertion effects. On adeno-associated virus (AAV) capsid variants, we show that relative differences in sequence log-likelihoods correlate with the experimentally measured insertion effects from Bryant et al. [21]. As shown in Table B.8, both GVP-GNN and GVP-Transformer outperform the sequence-only zero-shot prediction baseline ESM-1v [88]. When evaluating on subsets of sequences increasingly further away from the wild-type (≥ 2, ≥ 3, and ≥ 8 mutations), the GVP-GNN-large and GVP-Transformer models trained with predicted structures have increasing advantages compared to GVP-GNN trained without predicted structures.

3.3 Related Work

Structure-based protein sequence design Early work on design of protein sequences studied the packing of amino acid side chains to fill the interior space of predetermined backbone structures, either for a fixed backbone conformation [125, 26, 29], or with flexibility in the backbone conformation [47]. Since then, the Rosetta energy function [3] has become an established approach for structure-based sequence design. An alternative non-parametric approach involves decomposing the library of known structures into common sequence-structure motifs [149].

Early machine learning approaches in structure-based protein sequence design used fragment and energy-based global features derived from structures [82, 92]. More recently, convolution-based deep learning methods have also been applied to predict amino acid propensities given the surrounding local structural environments [7, 19, 118, 81, 99, 148, 24, 137]. Another recent machine learning approach is to leverage structure prediction networks for sequence design. Anishchenko et al. [9] carried out Monte Carlo sampling in the sequence space to invert the trRosetta [145] structure prediction network for sequence design, while Norn et al. [91] backpropagated gradients through the trRosetta network.

Generative models of proteins The literature on structure-based generative models of protein sequences is the closest to our work. Ingraham et al. [61] introduced the formulation of fixed-backbone design as a conditional sequence generation problem, using invariant features with graph neural networks, modeling each amino acid as a node in the graph with edges connecting spatially adjacent amino acids. Jing et al. [67, 66] further improved graph neural networks for this task by developing architectures with translation- and rotation-equivariance to enable geometric reasoning, showing that GVP-GNN achieves higher native sequence recovery rates than Rosetta on TS50, a benchmark set of 50 protein chains. Strokach et al.

[126] trained graph neural networks for conditional generation with the masked language modeling objective, adding homologous sequences as data augmentation to training. Most recently, contemporary with our work, Dauparas et al. [28] improved an existing graph neural network [61] by combining additional input features and edge updates, and validated designed protein sequences through X-ray crystallography, cryoEM and functional studies. Also contemporary with our work, Yang, Zanichelli, and Yeh [147] showed that pretrained sequence-only protein masked language models can be combined with structure-based graph neural networks to improve inverse folding performance.

Recently models have been proposed to jointly generate structures and sequences. Anishchenko et al. [9] generate structures by optimizing sequences through the trRosetta structure prediction network to maximize their difference from a background distribution. The joint generation approach is also being explored in the setting of infilling partial structures. Contemporary to this work, Wang et al. [138] apply span masking to fine-tune the RosettaFold model [11] to perform infilling, although conditioning on both coordinates and amino acid identities instead of considering the inverse folding task. Also contemporary to this work, Jin et al. [65] develop a conditional generation model for jointly generating sequences and structures for antibody complementarity determining regions (CDRs), conditioned on framework region structures. Anand and Achim [5] introduced an equivariant denoising diffusion approach for jointly generating protein structures and sequences for infilling, loop completion, and beyond.

So far there has been little work on generative models of structures directly. Interesting examples include Anand and Huang [6] who model fixed-length protein backbones with generative adversarial networks (GANs) via pairwise distance matrices, and Eguchi et al. [36] who generate antibody structures with variational autoencoders (VAEs).

Language models A large body of work has focused on modeling the sequences in individual protein families. Shin et al. [117] show that protein-specific autoregressive sequence models trained on related proteins can predict point mutation and indel effects and design functional nanobodies. Trinquier et al. [130] also studied protein-specific autoregressive models for sequence generation.

Recently language models have been proposed for modeling large scale databases of protein sequences rather than families of related sequences. Examples include [13, 4, 51, 101, 86, 38, 105, 102]. Meier et al. [88] found that the log-likelihoods of large protein language models predict mutational effects. Madani et al. [84] study an autoregressive sequence model conditioned on functional annotations and show it can generate functional proteins.

Structure-agnostic protein sequence design We point the reader to Wu et al. [144] for a review of the many machine learning-based sequence design approaches that do not explicitly model protein structures. Additionally, as an alternative to sequence generation models, model-guided algorithms design sequences based on predictive models as oracles [146, 8, 20, 120].

Back-translation For machine translation (MT) in NLP, Sennrich, Haddow, and Birch [113] studied how to leverage large amounts of monolingual data in the target language, a setting that parallels the situation we consider with protein sequences (the target language in our case). Sennrich et al. found it most effective to generate synthetic source sentences by performing the backwards translation from the target sentence, i.e. back-translation. This parallels the approach we take of predicting structures for sequence targets that have unknown structures. Edunov et al. [34] further investigated back-translation for large-scale language models.

3.4 Conclusions

While there are billions of protein sequences in the largest sequence databases, the number of available experimentally determined structures is on the order of hundreds of thousands, imposing a limit on generative methods that learn from protein structure data. In this work, we explored whether predicted structures from recent deep learning methods can be used in tandem with experimental structures to train models for protein design.

To this end, we generated structures for 12 million UniRef50 sequences using AlphaFold2. As a result of training with this data we observe improvements in perplexity and sequence recovery by substantial margins, and demonstrate generalization to longer protein complexes, to proteins in multiple conformations, and to zero-shot prediction for mutation effects on binding affinity and AAV packaging. These results highlight that in addition to the geometric inductive biases which have been the major focus for work on inverse-folding to date, finding ways to leverage more sources of training data is an equally important path to improved modeling capabilities.

Contemporary with our work, the AlphaFold Protein Structure Database [133] is rapidly growing, featuring 1 million predicted structures as of June 2022. Structure-based protein design models will likely continue to benefit from this new data source as the coverage expands to encompass more of the structural universe. Additionally, with the recent progress in structure prediction for multi-chain protein complexes [39, 60], predicted complex structures could be another valuable source of data for learning protein-protein interactions.

We also take initial steps toward more general structure-conditional protein design tasks. By integrating backbone span masking into the inverse folding task and using a sequence-to-sequence transformer, reasonable sequence predictions can be achieved for short masked spans.

If ways can be found to continue to leverage predicted structures for generative models of proteins, it may be possible to create models that learn to design proteins from an expanded universe of the billions of natural sequences whose structures are currently unknown.

Chapter 4

Perspectives on Conditional Generative Models

The inverse folding model presented in Chapter 3 is an example of a *conditional generative model*—one that can directly generate protein sequences based on a condition. This chapter aims to give a broader perspective of conditional generative models in the context of protein engineering and protein design, oriented towards an audience newer to machine learning.

To generate proteins from a conditional generative model, we specify a desired *condition* as input. The condition is a property of the protein, such as the backbone structure, melting temperature, or cellular localization. Once this condition is specified, the model allows us to *sample*—that is to generate stochastically—proteins that are likely to satisfy the condition. We might sample the protein in the form of a backbone structure, or a sequence, or both,

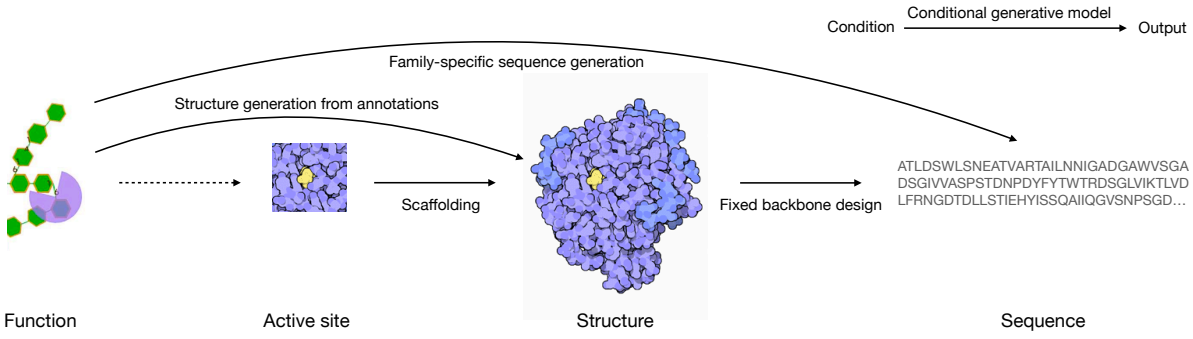

Figure 4.1: Conditional generative models can condition on, and generate, different types of entities in protein engineering, such as generating sequence from structure (*i. e.,* fixed backbone sequence design), sequence from function, and structure from active site geometry (*i. e.,* scaffolding).

for example. Technically, we are sampling a protein, X, from a *conditional* probability distribution, $p_\theta(X \mid Y = y)$, where θ are the learned parameters of the conditional generative model, and Y is the condition. Each setting, $Y = y$, corresponds to a conditional probability distribution over X.

Within protein engineering, conditional generative models have gained attention primarily for their use in generating sequences conditioned on fixed backbones, known as fixed backbone sequence design or "inverse folding" [61, 68, 28, 57]; and in generating backbone structure conditioned on the geometry of a known functional site—so-called "functional site scaffolding" [140, 62, 131]. There has also been work in generating sequences conditioned on a particular protein family [85]. See Figure 4.1 for some examples of the use of conditional generative models in protein engineering. For simplicity, we anchor our discussion on models that generate amino acid sequences or 3D structures, but much of our discussion applies equally to other types of modeling problems. Below, we discuss what types of problems are well-suited to being tackled by conditional generative models, touch upon their training and sampling procedures, and introduce two instantiations of conditional generative models in protein engineering for sequence and structure generation respectively.

4.1 When to use Conditional Generative Models?

As there are other ML approaches to sequence generation, it is helpful to first understand when it might be worthwhile to adopt a strategy of conditional generative modeling. First and foremost, conditional generative models are built to be general-purpose, in the sense that they aim to handle a variety of conditions, rather than being bespoke solutions for just one condition. For example, while a structure-based conditional generative model always takes 3D coordinates as inputs, it can in principle handle a wide variety of structures as input conditions. In contrast, one might instead devise a model that could only generate sequences that fold into the structure of a particular fold—a single condition by way of a specialized *unconditional generative model*. The generality of conditional generative models means that they correspondingly need to be trained on a wide variety of data that sufficiently encapsulates the intended use cases.

In comparison to unconditional generative models, one appeal of conditional generative models is that they need not be re-trained for every new condition. Moreover, by modeling all the different conditions together in the same model, conditional generative models can learn each conditional distribution in a way that "borrows" relevant information from "similar" conditions. Therefore, a particularly compelling setting arises when the exact condition of interest is rarely present in the training data, but related conditions are abundant. For example, we may not have examples of the exact backbone structure, but a conditional generative model can learn from other similar backbones, or backbones that have parts that are similar. By leveraging the data for all conditions in one model, a conditional generative model may also be better able to overcome measurement noise inherent to many biological data, as compared to unconditional generative models. More specialized, unconditional

approaches for sequence generation might work just as well as general-purpose conditional generative models when there are abundant data examples for the one condition of interest. For example, when there are many evolutionarily related homologs for the protein of interest, a specialized "evolutionary" model for this protein family could be a fruitful strategy [109, 49].

4.2 Training Conditional Generative Models

Machine learning models come with parameters to be learned during the training procedure. When we train the model on data, we select ("learn") one specific set of parameter values that is best suited to the training data. To learn the parameters of a model, be it a neural network or a classical statistical model, we must define a *loss function* which tells us how much we "lose" by using a particular set of parameters—that is, how "inappropriate" a parameter setting is for the training data. We then find the most appropriate parameter values—those that minimize the loss. A classical objective function from statistics is that of the likelihood of the observed data, which, when optimized, yields *maximum likelihood estimation* (MLE) of the parameters. MLE has the nice statistical property that, assuming the model class encompasses the true data distribution, then as we get increasingly more training data from that distribution, the estimated parameters become closer and closer to the true parameters. Due to this property of *consistency* and other favorable properties in statistical theory, a broad range of ML models are typically trained according to the MLE principle. When applied to conditional generative models, the MLE principle maximizes the average *conditional* log-likelihoods of the observed data. In deep learning, MLE is often described as minimizing the *cross-entropy* between the observed distribution and the predicted distribution—an information theoretic notion to measure the difference between two distributions. To minimize the loss, we start of with some initial setting of the parameters, typically at random, and then iteratively refine them using techniques from numerical optimization, mostly gradient descent, until they converge to a steady-state parameter setting.

The choice of training data directly influences the loss function, and hence, the resulting model. This choice is a *subjective* decision, and a very important one. In choosing our training data we express our beliefs about which sequence distribution we seek to learn. For example, suppose we train a structure-conditional generative model for generating sequences with data from the Protein Data Bank (PDB). And suppose there exist two sequences, which according to the principles of physics, both fold up equally well into a particular structure condition. Then our trained model will be more likely to generate the sequence that "appears" most similar to PDB sequences. Sometimes we may chose to *weight* each training example to coerce the trained model to represent a different distribution from the training data. Or, instead of re-weighting training examples, we may choose to exclude some. For example, we might group together individual training data (*i. e.,* cluster them) and then only train on one or a few representatives from each cluster, so that the model is not biased towards larger protein clusters. Note, however, that it is almost always impossible to

obtain the training data that expresses our modeling goals. For example, even when trying to approximate the distribution of naturally occurring protein sequences, we are subject to the whims of science that has had some proteins sequenced, and others not.

4.3 Sampling from Conditional Generative Models

Sampling is the statistical term that refers to generating sequences (or other entities such as structures), x, according to the conditional distribution $p_\theta(X \mid Y = y)$ for a given condition y, written $x \sim p_\theta(X \mid Y = y)$. For example, in fixed backbone sequence design, we sample sequences conditioned on a backbone structure. In functional site scaffolding, we sample entire protein backbone structures conditioned on the functional site structure. Focusing on the example of sampling sequences: as there may be numerous sequences that satisfy a given condition, and also uncertainty in how well the condition is satisfied by each sequence, samples from the probability distribution are typically non-deterministic (*i. e.,* stochastic). Sequences that are more likely to satisfy the condition will have higher (conditional) likelihood, and are more likely to be sampled. However, by chance alone, we may sample sequences that do not satisfy the condition at all.

Naively, the conditional distribution over all possible outputs can be unwieldy to model, and difficult to draw samples from. For conditional generative models over sequences, there are 20^L possible amino acid sequences that can be generated for a protein of length L. For conditional generative models over 3D structures, there is an even larger space of possible generated structure outputs. Different strategies can be used to break down the modeling and sampling problem into smaller, more manageable parts. For example, we can *factor* the conditional probability distribution—that is, decompose the likelihood into separate factors, each of which is manageable, and all of which can be easily be combined together as a product. In the next section, we discuss the commonly used autoregressive factorization of conditional generative models for sequences. An alternative approach for more easily generating samples from large output spaces is to use a class of models wherein we break down the sampling approach into an iterative process. In this class of models each sample depends on the previous sample, in a relatively simple way, such that after many iterations, we can obtain good quality samples. We discuss this diffusion model strategy in the section on generating structures.

When sampling a set of sequences to be synthesized and assayed in the laboratory, we may be interested in controlling the diversity of those sequences so as to not waste our budget with highly similar sequences. Broadly, in many ML application areas, diversity is controlled by modulating the *statistical entropy* of the distribution. Sampling from a low-entropy distribution will tend to generate the same sequences over and over. Sampling from a high-entropy distribution will be much less likely to do so. Although by virtue of training a model to learn its parameters, we fix the entropy of the model, for many ML models, a *post hoc* procedure can be used to control the entropy to be higher or lower. To do so, we use a single scalar, a "temperature" parameter, T, which is introduced to the model after it

has been trained. Implicitly, without this temperature parameter, the model already has the default value $T = 1$, which thus serves as a reference point for the entropy of the training data (according to the model). The higher the temperature, the higher the entropy. As we increase the diversity of a set of sampled sequences, we correspondingly reduce the average likelihood of those sequences under the model, indicative of an inherent trade-off between these. In other words, if we want more cards to bet with, then we're going to have to go down into the lower cards of the deck, having run out of Aces, Kings and Queens.

Next, we discuss two instantiations of conditional generative models for protein engineering in more detail. One is for sequence generation, and the other, for structure generation.

4.4 Conditional Generative Models for Sequences

As mentioned in the last section, modeling and sampling distributions over large output spaces are difficult to directly work with, requiring strategies such as factorization to make them more manageable. One particularly effective factorization strategy for sampling sequences is to use a *chain-rule factorization*, which models each amino acid based on all the previous amino acids (according to, for example, the sequential ordering on the backbone chain) and on the input condition. Such a factorization, by itself, doesn't make any assumptions about the distribution that can be captured by the model. Generative models that use this factorization are referred to as *autoregressive models*.

For any sequence, x with L positions, $x = x_1 x_2 \dots x_L$, the chain rule factors the conditional sequence likelihood, $p_\theta(X_1 = x_1, X_2 = x_2, \dots, X_L = x_L \mid Y = y)$, into the following product: the likelihood of $X_1 = x_1$, multiplied by the likelihood of $X_2 = x_2$ conditioned on $X_1 = x_1$, multiplied by the likelihood of $X_3 = x_3$ conditioned on $X_1 = x_1$ and $X_2 = x_2$, ..., and so on and so forth. One limitation of autoregressive models is that naively the model can only be used to generate sequences in the chosen sequential ordering. To address this limitation, one could explicitly train models with different orderings [28]. The factorization of a conditional distribution need not (and typically does not) reflect any causal mechanism so any ordering is mathematically valid, although different choices of ordering of the variables will generally lead to different models [130]. A common alternative to autoregressive models in ML are "masked language models" which can be better at capturing the rich interplay between positions in a sequence. However, the likelihood of such models cannot be readily computed, nor can we easily sample from them.

Modern day autoregressive conditional generative models typically employ a neural network architecture to parameterize their class of probability distributions. Consequently, the same neural network considerations that emerge in other general ML settings, similarly emerge here. For example, we must consider architecture choices, such as whether or not to use convolution, attention, and how large of a network to use—how many layers, and the width of each layer. The neural network architecture could be graph-based and could encode certain symmetries of the physical world, such as that a protein is the same protein no matter how we rotate it in space (*e. g.,* equivariant to 3D rotations). As these neural net-

work modeling considerations depend on the specific problem settings, there is no universal solution.

Having said that, in practice, we often use an encoder-decoder architecture for autoregressive conditional generative models. The *encoder* network encodes the input condition y into a "hidden"/latent representation, and the *decoder* network decodes the hidden representation into a conditional sequence distribution. Within autoregressive conditional generative models, special care is required in constructing the decoder (*e.g.,* with "causal masking" and "teacher forcing"), since the decoding proceeds in a step-wise fashion for one amino acid at a time, each time considering the previous amino acids already generated. Beside the encoder-decoder architecture, decoder-only architectures have also been used for conditional generation, whereby sequences are conditioned directly on a label for the protein family [85].

4.5 Conditional Generative Models for Structures

While protein sequences consist of a finite vocabulary of 20 amino acids, the 3D coordinates of protein structures are real-values and therefore introduce a different set of modeling challenges. While there are many existing approaches to generating protein structures [5, 80, 35, 52, 139, 135, 103], recently diffusion models have emerged as a powerful class of models for both unconditional and conditional generation [131, 140, 62, 5]. Instead of estimating a (conditional) probability distribution over structures, *diffusion models* estimate how a (conditional) probability distribution behaves in local neighborhoods of structure space. That is, how the likelihood changes as we perturb a protein structure, at any point in structure space. Formally, we learn the *gradient* of the probability distribution rather than the distribution itself. The probability distribution can implicitly be recovered from the gradient information. At train time, we iteratively "diffuse" a protein structure in the training data by gradually adding random noise to it, until it is unrecognizable. We train a neural network to learn how to reverse the diffusion process, step-by-step, such that we can go from the unrecognizable structure, to the original, well-formed protein structure. To generate structures from a trained diffusion model, we use the reverse diffusion process, starting with a random structure, and incrementally adjusting the set of 3D coordinates to "move" towards more likely coordinates. While this may seem analogous to following a force field in molecular dynamics simulations, the diffusion process in diffusion models does not necessarily carry physical meaning.

There are two main approaches to enable diffusion models to be *conditional* diffusion models, and both have been used in protein structure generation. In "classifier-free guidance" [131, 140], the neural network that reverses the diffusion takes in a "condition tag" that enables it to tailor the process accordingly. Such models must be trained with the conditions as the conditions are tied into the neural network. In contrast, in "classifier-guided" conditioning [62], the conditioning does not happen in the neural network, rather, the neural network is paired with an external classifier that has been separately trained to classify whether a 3D structure is likely to satisfy the condition of interest. Consequently, the diffu-

sion process, and the guidance, are conveniently decoupled. In particular, classifier-guided conditioning allows the same unconditional diffusion model to be easily re-purposed for a wide range of conditions without re-training. Of course the quality of the conditional generation depends on the quality of the classifier. In the reverse diffusion process, the external classifier is used in a technically precise way (that leverages Bayes' rule) to enable the diffusion process to sample from the desired conditional distribution. Using a classifier-guided approach has enabled conditional generative models to accommodate a wide range of different input condition modalities in a plug-and-play manner, including: secondary structures, coarse contact maps, partial structural motifs, symmetry, and biochemical properties [62].

Although in principle, it is possible to jointly model/generate the structure and sequence together in diffusion models, this remains a challenging task. Currently, diffusion models for generating protein structures typically require a second modeling step to find sequences for the backbone structures.

4.6 Alternative Use: Evaluating Candidates by Conditional Likelihoods

Although named "generative" models, many conditional generative models can also be used to evaluate existing candidate sequences (or candidate structures) by their conditional likelihoods. The *conditional likelihood*, $p_\theta(X = x \mid Y = y)$, can be used to rank how likely a sequence, x, is to satisfy the given condition y, in comparison to other possible sequences. We can only obtain a ranking, rather than an absolute probability, for the following reasons. The conditional likelihood does not take into account whether the condition itself is inherently easy or difficult to satisfy. We would have to enumerate all possible sequences to "normalize" the likelihoods to get the desired probability.

Nevertheless, these conditional likelihoods can be useful for the relative ranking of candidate sequences. For example, from a conditional generative model for backbone-conditional sequence generation, we may evaluate say 1,000 different candidate sequences to ask which one is our best bet for forming a specified backbone. To do so, we compare the conditional likelihoods of each sequence conditioned on the same backbone; all else being equal, those with higher conditional likelihoods are more likely to adhere to the input condition. For a fixed backbone, the relative ranking of the conditional likelihoods of these sequences has been shown to correlate well with experimental measurements of stability [61, 68, 57]. Despite that the models were not explicitly trained with stability measurements, stability can be implicitly learned because sequences corresponding to more stable proteins were more likely to appear in the training data—highlighting how the choice of data infiltrates the modeling process, for good or for bad.

4.7 Final Considerations

Unlike the case for generated images and texts, the average human cannot look at a generated protein sequence or structure and decide whether it meets their needs or not. Rather, we require expensive and time-consuming experimental work to validate any candidates. Hence generative biology is in a much more difficult position than generative modeling for images and texts. But the potential rewards, for engineering therapeutics and enzymes, for example, are, arguably, correspondingly larger.

A potential danger of conditional generative models is a belief that once we build such a model and sample from it, we will obtain the proteins that we set out to find. In practice, these models are only as good as the parts that comprise it, including the data it was trained on and the auxiliary models used in conditioning.

It's worth noting that while ML has rapidly transformed the field of structure prediction, there remains a large amount of work to be done to reliably predict protein functions of almost any kind. Consequently, it will generally be more difficult to build accurate conditional generative models for specific functions, rather than structures. Functions that are largely determined by macroscopic structure, such as binding, are more in reach, while those requiring very precise active sites, such as enzyme activity, remain much less predictable. The discrepancy in our success predicting function versus structure is due in part to the relative dearth of data for protein function, while protein structures are now quite numerous. Efforts are underway to generate proteins by conditioning on functional annotations, building upon natural language as a "unified" interface for describing protein function. However, annotations can be unreliable, and may be less amenable to data sharing across similar annotations than more fundamental physical descriptors. For many important protein functions of interest, experimental screening technologies will remain a critical part of the protein engineering workflow.

Bibliography

[1] Ivan A Adzhubei et al. "A method and server for predicting damaging missense mutations". In: *Nature Methods* 7.4 (2010), pp. 248–249.

[2] Amirali Aghazadeh et al. "Epistatic Net allows the sparse spectral regularization of deep neural networks for inferring fitness functions". In: *Nature communications* 12.1 (2021), p. 5225.

[3] Rebecca F Alford et al. "The Rosetta all-atom energy function for macromolecular modeling and design". In: *Journal of Chemical Theory and Computation* 13.6 (2017), pp. 3031–3048.

[4] Ethan C Alley et al. "Unified rational protein engineering with sequence-based deep representation learning". In: *Nature Methods* 16.12 (2019), pp. 1315–1322.

[5] Namrata Anand and Tudor Achim. *Protein Structure and Sequence Generation with Equivariant Denoising Diffusion Probabilistic Models*. 2022. arXiv: 2205.15019.

[6] Namrata Anand and Possu Huang. "Generative modeling for protein structures". In: *Advances in neural information processing systems* 31 (2018).

[7] Namrata Anand-Achim et al. "Protein sequence design with a learned potential". In: *Biorxiv* (2021), pp. 2020–01.

[8] Christof Angermueller et al. "Model-based reinforcement learning for biological sequence design". In: *International conference on learning representations*. 2019.

[9] Ivan Anishchenko et al. "De novo protein design by deep network hallucination". In: *Nature* 600.7889 (2021), pp. 547–552.

[10] Carlos L Araya et al. "A fundamental protein property, thermodynamic stability, revealed solely from large-scale measurements of protein function". In: *Proceedings of the National Academy of Sciences* 109.42 (2012), pp. 16858–16863.

[11] Minkyung Baek et al. "Accurate prediction of protein structures and interactions using a three-track neural network". In: *Science* 373.6557 (2021), pp. 871–876. DOI: 10.1126/science.abj8754.

[12] Pierre Barrat-Charlaix, Matteo Figliuzzi, and Martin Weigt. "Improving landscape inference by integrating heterogeneous data in the inverse Ising problem". In: *Scientific Reports* 6.1 (2016), pp. 1–9.

[13] Tristan Bepler and Bonnie Berger. "Learning protein sequence embeddings using information from structure". In: *arXiv preprint arXiv:1902.08661* (2019).

[14] Helen M Berman et al. "The protein data bank". In: *Nucleic acids research* 28.1 (2000), pp. 235–242.

[15] Julian Besag. "Statistical analysis of non-lattice data". In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 24.3 (1975), pp. 179–195.

[16] H Kaspar Binz, Patrick Amstutz, and Andreas Plückthun. "Engineering novel binding proteins from nonimmunoglobulin domains". In: *Nature Biotechnology* 23.10 (2005), pp. 1257–1268.

[17] Surojit Biswas et al. "Low-N protein engineering with data-efficient deep learning". In: *Nature Methods* 18.4 (2021), pp. 389–396.

[18] Mathieu Blondel et al. "Fast differentiable sorting and ranking". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 950–959.

[19] Wouter Boomsma and Jes Frellsen. "Spherical convolutions and their application in molecular modelling". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/1113d7a76ffceca1bb350bfe145467c6-Paper.pdf.

[20] David Brookes, Hahnbeom Park, and Jennifer Listgarten. "Conditioning by adaptive sampling for robust design". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 773–782.

[21] Drew H Bryant et al. "Deep diversification of an AAV capsid protein by machine learning". In: *Nature Biotechnology* (2021), pp. 1–6.

[22] Emidio Capriotti, Piero Fariselli, and Rita Casadio. "I-Mutant2. 0: predicting stability changes upon mutation from the protein sequence or structure". In: *Nucleic Acids Research* 33.suppl_2 (2005), W306–W310.

[23] Martin Chalfie et al. "Green fluorescent protein as a marker for gene expression". In: *Science* 263.5148 (1994), pp. 802–805.

[24] Sheng Chen et al. "To improve protein sequence profile prediction through image captioning on pairwise residue distance map". In: *Journal of chemical information and modeling* 60.1 (2019), pp. 391–399.

[25] Ryan R Cheng et al. "Toward rationally redesigning bacterial two-component signaling systems using coevolutionary information". In: *Proceedings of the National Academy of Sciences* 111.5 (2014), E563–E571.

[26] Bassil I Dahiyat and Stephen L Mayo. "Probing the role of packing specificity in protein design". In: *Proceedings of the National Academy of Sciences* 94.19 (1997), pp. 10172–10177.

[27] Christian Dallago et al. "FLIP: Benchmark tasks in fitness landscape inference for proteins". In: *bioRxiv* (2021).

[28] Justas Dauparas et al. "Robust deep learning based protein sequence design using ProteinMPNN". In: *bioRxiv* (2022). eprint: https://www.biorxiv.org/content/early/2022/06/04/2022.06.03.494563.full.pdf. URL: https://www.biorxiv.org/content/early/2022/06/04/2022.06.03.494563.

[29] William F DeGrado, Daniel P Raleigh, and Tracey Handel. "De novo protein design: what are we learning?" In: *Current Opinion in Structural Biology* 1.6 (1991), pp. 984–993.

[30] Yves Dehouck et al. "PoPMuSiC 2.1: a web server for the estimation of protein stability changes upon mutation and sequence optimality". In: *BMC Bioinformatics* 12.1 (2011), pp. 1–12.

[31] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 4171–4186.

[32] Jennifer A Doudna and Emmanuelle Charpentier. "The new frontier of genome engineering with CRISPR-Cas9". In: *Science* 346.6213 (2014).

[33] Sean R. Eddy. "Profile hidden Markov models." In: *Bioinformatics (Oxford, England)* 14.9 (1998), pp. 755–763.

[34] Sergey Edunov et al. "Understanding back-translation at scale". In: *arXiv preprint arXiv:1808.09381* (2018).

[35] Raphael R Eguchi, Christian A Choe, and Po-Ssu Huang. "Ig-VAE: Generative modeling of protein structure by direct 3D coordinate generation". In: *PLoS computational biology* 18.6 (2022), e1010271.

[36] Raphael R Eguchi et al. "Ig-VAE: generative modeling of immunoglobulin proteins by direct 3D coordinate generation". In: *bioRxiv* (2020).

[37] Ahmed Elnaggar et al. "ProtTrans: Towards Cracking the Language of Life's Code Through Self-Supervised Deep Learning and High Performance Computing". In: (2020). Preprint at https://www.biorxiv.org/content/10.1101/2020.07.12.199554v1.

[38] Ahmed Elnaggar et al. "ProtTrans: Towards Cracking the Language of Lifes Code Through Self-Supervised Deep Learning and High Performance Computing". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3095381.

[39] Richard Evans et al. "Protein complex prediction with AlphaFold-Multimer". In: *bioRxiv* (2022). eprint: https://www.biorxiv.org/content/early/2022/03/10/2021.10.04.463034.full.pdf. URL: https://www.biorxiv.org/content/early/2022/03/10/2021.10.04.463034.

[40] Clara Fannjiang and Jennifer Listgarten. "Autofocused oracles for model-based design". In: *Advances in Neural Information Processing Systems* 33 (2020).

[41] Matteo Figliuzzi et al. "Coevolutionary landscape inference and the context dependence of mutations in beta-lactamase TEM-1". In: *Molecular Biology and Evolution* 33.1 (2016), pp. 268–280.

[42] Robert D Finn, Jody Clements, and Sean R Eddy. "HMMER web server: interactive sequence similarity searching". In: *Nucleic Acids Research* 39.Web Server issue (2011), W29.

[43] Sam Gelman, Philip A Romero, and Anthony Gitter. "Neural networks to learn protein sequence-function relationships from deep mutational scanning data". In: (2020). Preprint at https://www.biorxiv.org/content/10.1101/2020.10.25.353946v2.

[44] Alexander G Georgiev. "Interpretable numerical descriptors of amino acid space". In: *Journal of Computational Biology* 16.5 (2009), pp. 703–723.

[45] Vladimir Gligorijevic et al. "Function-guided protein design by deep manifold sampling". In: *bioRxiv* (2021).

[46] Vanessa E Gray et al. "Quantitative missense variant effect prediction using large-scale mutagenesis data". In: *Cell Systems* 6.1 (2018), pp. 116–124.

[47] Pehr B Harbury et al. "High-resolution protein design with backbone freedom". In: *Science* 282.5393 (1998), pp. 1462–1467.

[48] Moritz Hardt and Benjamin Recht. *Patterns, predictions, and actions: A story about machine learning*. https://mlstory.org, 2021. arXiv: 2102.05242 [cs.LG].

[49] Alex Hawkins-Hooker et al. "Generating functional protein variants with variational autoencoders". en. In: *PLoS Comput. Biol.* 17.2 (Feb. 2021), e1008736.

[50] Roger Heim and Roger Y Tsien. "Engineering green fluorescent protein for improved brightness, longer wavelengths and fluorescence resonance energy transfer". In: *Current Biology* 6.2 (1996), pp. 178–182.

[51] Michael Heinzinger et al. "Modeling aspects of the language of life through transfer-learning protein sequences". In: *BMC bioinformatics* 20.1 (2019), pp. 1–17.

[52] Brian Hie et al. "A high-level programming language for generative protein design". In: *bioRxiv* (2022), pp. 2022–12.

[53] Thomas A Hopf et al. "Mutation effects predicted from sequence co-variation". In: *Nature Biotechnology* 35.2 (2017), pp. 128–135.

[54] Viktor Hornak et al. "Comparison of multiple Amber force fields and development of improved protein backbone parameters". In: *Proteins: Structure, Function, and Bioinformatics* 65.3 (2006), pp. 712–725.

[55] Jeremy Howard and Sebastian Ruder. "Universal Language Model Fine-tuning for Text Classification". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 328–339.

[56] Thomas Hrabe et al. "PDBFlex: exploring flexibility in protein structures". In: *Nucleic acids research* 44.D1 (2016), pp. D423–D428.

[57] Chloe Hsu et al. "Learning inverse folding from millions of predicted structures". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 8946–8970.

[58] Patrick D Hsu, Eric S Lander, and Feng Zhang. "Development and applications of CRISPR-Cas9 for genome engineering". In: *Cell* 157.6 (2014), pp. 1262–1278.

[59] Po-Ssu Huang, Scott E Boyken, and David Baker. "The coming of age of de novo protein design". In: *Nature* 537.7620 (2016), pp. 320–327.

[60] Ian R Humphreys et al. "Computed structures of core eukaryotic protein complexes". In: *Science* 374.6573 (2021).

[61] John Ingraham et al. "Generative Models for Graph-Based Protein Design". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[62] John Ingraham et al. "Illuminating protein space with a programmable generative model". In: *bioRxiv* (2022), pp. 2022–12.

[63] Justina Jankauskaitė et al. "SKEMPI 2.0: an updated benchmark of changes in protein–protein binding energy, kinetics and thermodynamics upon mutation". In: *Bioinformatics* 35.3 (2019), pp. 462–469.

[64] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446.

[65] Wengong Jin et al. "Iterative refinement graph neural network for antibody sequence-structure co-design". In: *arXiv preprint arXiv:2110.04624* (2021).

[66] Bowen Jing et al. "Equivariant Graph Neural Networks for 3D Macromolecular Structure". In: *Proceedings of the International Conference on Machine Learning* (2021).

[67] Bowen Jing et al. "Learning from Protein Structure with Geometric Vector Perceptrons". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[68] Bowen Jing et al. "Learning from Protein Structure with Geometric Vector Perceptrons". In: *International Conference on Learning Representations*. 2021.

[69] Hyun Joo, Zhanglin Lin, and Frances H Arnold. "Laboratory evolution of peroxide-mediated cytochrome P450 hydroxylation". In: *Nature* 399.6737 (1999), pp. 670–673.

[70] Mandar Joshi et al. "Spanbert: Improving pre-training by representing and predicting spans". In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 64–77.

[71] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* (2021), pp. 1–11.

[72] Martin Karplus and John Kuriyan. "Molecular dynamics and protein function". In: *Proceedings of the National Academy of Sciences* 102.19 (2005), pp. 6679–6685.

[73] S Kawashima et al. "AAindex: amino acid index database, progress report 2008." In: *Nucleic Acids Research* 36.Database issue (2007), pp. D202–5.

[74] Michel van Kempen et al. "Foldseek: fast and accurate protein structure search". In: *bioRxiv* (2022).

[75] Patrick Kunzmann and Kay Hamacher. "Biotite: a unifying open source computational biology framework in Python". In: *BMC bioinformatics* 19.1 (2018), pp. 1–8.

[76] Jun Lan et al. "Structure of the SARS-CoV-2 spike receptor-binding domain bound to the ACE2 receptor". In: *Nature* 581.7807 (2020), pp. 215–220.

[77] Robert A Langan et al. "De novo design of bioactive protein switches". In: *Nature* 572.7768 (2019), pp. 205–210.

[78] Benjamin Leader, Quentin J Baca, and David E Golan. "Protein therapeutics: a summary and pharmacological classification". In: *Nature Reviews Drug discovery* 7.1 (2008), pp. 21–39.

[79] Jonathan Lees et al. "Gene3D: a domain-based resource for comparative genomics, functional annotation and protein network analysis". In: *Nucleic acids research* 40.D1 (2012), pp. D465–D471.

[80] Alex J Li et al. "Neural network-derived Potts models for structure-based protein design using backbone atomic coordinates and tertiary motifs". In: *Protein Science* 32.2 (2023), e4554.

[81] Bian Li et al. "Predicting changes in protein thermodynamic stability upon point mutation with deep 3D convolutional neural networks". In: *PLoS computational biology* 16.11 (2020), e1008291.

[82] Zhixiu Li et al. "Direct prediction of profiles of sequences compatible with a protein structure by neural networks with fragment-based local and energy-based nonlocal profiles". In: *Proteins: Structure, Function, and Bioinformatics* 82.10 (2014), pp. 2565–2573.

[83] Zeming Lin et al. "Evolutionary-scale prediction of atomic-level protein structure with a language model". In: *Science* 379.6637 (2023), pp. 1123–1130.

[84] Ali Madani et al. "Deep neural language modeling enables functional protein generation across families". In: *bioRxiv* (2021).

[85] Ali Madani et al. "Large language models generate functional protein sequences across diverse families". In: *Nature Biotechnology* (2023), pp. 1–8.

[86] Ali Madani et al. "Progen: Language modeling for protein generation". In: (2020). Preprint at https://www.biorxiv.org/content/10.1101/2020.03.07.982272v2.

[87] Jaclyn K Mann et al. "The fitness landscape of HIV-1 gag: advanced modeling approaches and validation of model predictions by in vitro testing". In: *PLoS Computational Biology* 10.8 (2014), e1003776.

[88] Joshua Meier et al. "Language models enable zero-shot prediction of the effects of mutations on protein function". In: *Advances in Neural Information Processing Systems* 34 (2021).

[89] Daniel Melamed et al. "Deep mutational scanning of an RRM domain of the Saccharomyces cerevisiae poly (A)-binding protein". In: *RNA* 19.11 (2013), pp. 1537–1551.

[90] Milot Mirdita et al. "Uniclust databases of clustered and deeply annotated protein sequences and alignments". In: *Nucleic acids research* 45.D1 (2017), pp. D170–D176.

[91] Christoffer Norn et al. "Protein sequence design by conformational landscape optimization". In: *Proceedings of the National Academy of Sciences* 118.11 (2021).

[92] James O'Connell et al. "SPIN2: Predicting sequence profiles from protein structures using deep neural networks". In: *Proteins: Structure, Function, and Bioinformatics* 86.6 (2018), pp. 629–633.

[93] C Anders Olson, Nicholas C Wu, and Ren Sun. "A comprehensive biophysical description of pairwise epistasis throughout an entire protein domain". In: *Current Biology* 24.22 (2014), pp. 2643–2651.

[94] Christine A Orengo et al. "CATH–a hierarchic classification of protein domain structures". In: *Structure* 5.8 (1997), pp. 1093–1109.

[95] Myle Ott et al. "fairseq: A fast, extensible toolkit for sequence modeling". In: *arXiv preprint arXiv:1904.01038* (2019).

[96] Jakub Otwinowski, David M McCandlish, and Joshua B Plotkin. "Inferring the shape of global epistasis". In: *Proceedings of the National Academy of Sciences* 115.32 (2018), E7550–E7558.

[97] Loredano Pollegioni, Ernst Schonbrunn, and Daniel Siehl. "Molecular basis of glyphosate resistance–different approaches through protein engineering". In: *The FEBS journal* 278.16 (2011), pp. 2753–2766.

[98] Simon C Potter et al. "HMMER web server: 2018 update". In: *Nucleic acids research* 46.W1 (2018), W200–W204.

[99] Yifei Qi and John ZH Zhang. "DenseCPD: improving the accuracy of neural-network-based computational protein sequence design with DenseNet". In: *Journal of Chemical Information and Modeling* 60.3 (2020), pp. 1245–1252.

[100] Alfredo Quijano-Rubio et al. "De novo design of modular and tunable protein biosensors". In: *Nature* 591.7850 (2021), pp. 482–487.

[101] Roshan Rao et al. "Evaluating protein transfer learning with TAPE". In: *Advances in Neural Information Processing Systems*. 2019, pp. 9689–9701.

[102] Roshan Rao et al. "Msa transformer". In: *bioRxiv* (2021).

[103] Stephen A Rettie et al. "Cyclic peptide structure prediction and design using AlphaFold". In: *bioRxiv* (2023), pp. 2023–02.

[104] Adam J Riesselman, John B Ingraham, and Debora S Marks. "Deep generative models of genetic variation capture the effects of mutations". In: *Nature Methods* 15.10 (2018), pp. 816–822.

[105] Alexander Rives et al. "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences". In: *Proceedings of the National Academy of Sciences* 118.15 (2021).

[106] Alexander Rives et al. "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences". In: *Proceedings of the National Academy of Sciences* 118.15 (2021). DOI: 10.1073/pnas.2016239118.

[107] Gabriel J Rocklin et al. "Global analysis of protein folding using massively parallel design, synthesis, and testing". In: *Science* 357.6347 (2017), pp. 168–175.

[108] Philip A Romero, Andreas Krause, and Frances H Arnold. "Navigating the protein fitness landscape with Gaussian processes". In: *Proceedings of the National Academy of Sciences* 110.3 (2013), E193–E201.

[109] William P Russ et al. "An evolution-based model for designing chorismate mutase enzymes". In: *Science* 369.6502 (2020), pp. 440–445.

[110] Karen S Sarkisyan et al. "Local fitness landscape of the green fluorescent protein". In: *Nature* 533.7603 (2016), pp. 397–401.

[111] Joost Schymkowitz et al. "The FoldX web server: an online force field". In: *Nucleic Acids Research* 33.suppl_2 (2005), W382–W388.

[112] Andrew W Senior et al. "Improved protein structure prediction using potentials from deep learning". In: *Nature* 577.7792 (2020), pp. 706–710.

[113] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Improving neural machine translation models with monolingual data". In: *arXiv preprint arXiv:1511.06709* (2015).

[114] Zahra Shamsi, Matthew Chan, and Diwakar Shukla. "TLmutation: predicting the effects of mutations using transfer learning". In: *The Journal of Physical Chemistry B* 124.19 (2020), pp. 3845–3854.

[115] Amir Shanehsazzadeh, David Belanger, and David Dohan. "Is Transfer Learning Necessary for Protein Landscape Prediction?" In: (2020). Preprint at https://arxiv.org/abs/2011.03443.

[116] Hashem A Shihab et al. "Predicting the functional, molecular, and phenotypic consequences of amino acid substitutions using hidden Markov models". In: *Human Mutation* 34.1 (2013), pp. 57–65.

[117] Jung-Eun Shin et al. "Protein design and variant prediction using autoregressive generative models". In: *Nature communications* 12.1 (2021), pp. 1–11.

[118] Raghav Shroff et al. "Discovery of novel gain-of-function mutations guided by structure-based deep learning". In: *ACS synthetic biology* 9.11 (2020), pp. 2927–2935.

[119] Ngak-Leng Sim et al. "SIFT web server: predicting effects of amino acid substitutions on proteins". In: *Nucleic Acids Research* 40.W1 (2012), W452–W457.

[120] Sam Sinai et al. "AdaLead: A simple and robust adaptive greedy search algorithm for sequence design". In: (2020). Preprint at https://arxiv.org/abs/2010.02141.

[121] Lea M Starita et al. "Activity-enhancing mutations in an E3 ubiquitin ligase identified by high-throughput mutagenesis". In: *Proceedings of the National Academy of Sciences* 110.14 (2013), E1263–E1272.

[122] Tyler N Starr et al. "Deep mutational scanning of SARS-CoV-2 receptor binding domain reveals constraints on folding and ACE2 binding". In: *Cell* 182.5 (2020), pp. 1295–1310.

[123] Richard R Stein, Debora S Marks, and Chris Sander. "Inferring pairwise interactions from biological data using maximum-entropy probability models". In: *PLoS Computational Biology* 11.7 (2015), e1004182.

[124] Martin Steinegger et al. "HH-suite3 for fast remote homology detection and deep protein annotation". In: *BMC bioinformatics* 20.1 (2019), pp. 1–15.

[125] Arthur G Street and Stephen L Mayo. "Computational protein design". In: *Structure* 7.5 (1999), R105–R109.

[126] Alexey Strokach et al. "Fast and flexible protein design using deep graph neural networks". In: *Cell Systems* 11.4 (2020), pp. 402–411.

[127] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. "Covariate Shift Adaptation by Importance Weighted Cross Validation". In: *Journal of Machine Learning Research* 8 (2007), pp. 985–1005.

[128] Baris E Suzek et al. "UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches". In: *Bioinformatics* 31.6 (2015), pp. 926–932.

[129] Raphael J. L. Townshend et al. "ATOM3D: Tasks On Molecules in Three Dimensions". In: *CoRR* abs/2012.04035 (2020). arXiv: 2012.04035.

[130] Jeanne Trinquier et al. "Efficient generative modeling of protein sequences using simple autoregressive models". In: *arXiv preprint arXiv:2103.03292* (2021).

[131] Brian L Trippe et al. "Diffusion probabilistic modeling of protein backbones in 3D for the motif-scaffolding problem". In: *arXiv preprint arXiv:2206.04119* (2022).

[132] Iulia Turc et al. "Well-read students learn better: On the importance of pre-training compact models". In: *arXiv preprint arXiv:1908.08962* (2019).

[133] Mihaly Varadi et al. "AlphaFold Protein Structure Database: massively expanding the structural coverage of protein-sequence space with high-accuracy models". In: *Nucleic Acids Research* 50.D1 (Nov. 2021), pp. D439–D444. ISSN: 0305-1048. eprint: https://academic.oup.com/nar/article-pdf/50/D1/D439/43502749/gkab1061.pdf. URL: https://doi.org/10.1093/nar/gkab1061.

[134] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[135] Robert Verkuil et al. "Language models generalize beyond natural proteins". In: *bioRxiv* (2022), pp. 2022–12.

[136] Alexandra C Walls et al. "Structure, function, and antigenicity of the SARS-CoV-2 spike glycoprotein". In: *Cell* 181.2 (2020), pp. 281–292.

[137] Jingxue Wang et al. "Computational protein design with deep learning neural networks". In: *Scientific reports* 8.1 (2018), pp. 1–9.

[138] Jue Wang et al. "Deep learning methods for designing proteins scaffolding functional sites". In: *bioRxiv* (2021).

[139] Jue Wang et al. "Scaffolding protein functional sites using deep learning". In: *Science* 377.6604 (2022), pp. 387–394.

[140] Joseph L Watson et al. "Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models". In: *bioRxiv* (2022), pp. 2022–12.

[141] Bruce J Wittmann, Yisong Yue, and Frances H Arnold. "Informed training set design enables efficient machine learning-assisted directed protein evolution". In: *Cell Systems* (2021).

[142] Nicholas C Wu et al. "Adaptation in protein fitness landscapes is facilitated by indirect paths". In: *Elife* 5 (2016), e16965.

[143] Ruidong Wu et al. "High-resolution de novo structure prediction from primary sequence". In: *BioRxiv* (2022), pp. 2022–07.

[144] Zachary Wu et al. "Protein sequence design with deep generative models". In: *Current Opinion in Chemical Biology* 65 (2021), pp. 18–27.

[145] Jianyi Yang et al. "Improved protein structure prediction using predicted interresidue orientations". In: *Proceedings of the National Academy of Sciences* 117.3 (2020), pp. 1496–1503.

[146] Kevin K Yang, Zachary Wu, and Frances H Arnold. "Machine-learning-guided directed evolution for protein engineering". In: *Nature Methods* 16.8 (2019), pp. 687–694.

[147] Kevin K Yang, Niccolò Zanichelli, and Hugh Yeh. "Masked inverse folding with sequence transfer for protein representation learning". In: *bioRxiv* (2022).

[148] Yuan Zhang et al. "ProDCoNN: Protein design using a convolutional neural network". In: *Proteins: Structure, Function, and Bioinformatics* 88.7 (2020), pp. 819–829.

[149] Jianfu Zhou, Alexandra E Panaitiu, and Gevorg Grigoryan. "A general-purpose protein design framework based on mining sequence–structure relationships in known protein structures". In: *Proceedings of the National Academy of Sciences* 117.2 (2020), pp. 1059–1068.

Appendix A

Supplementary Information for Chapter 2

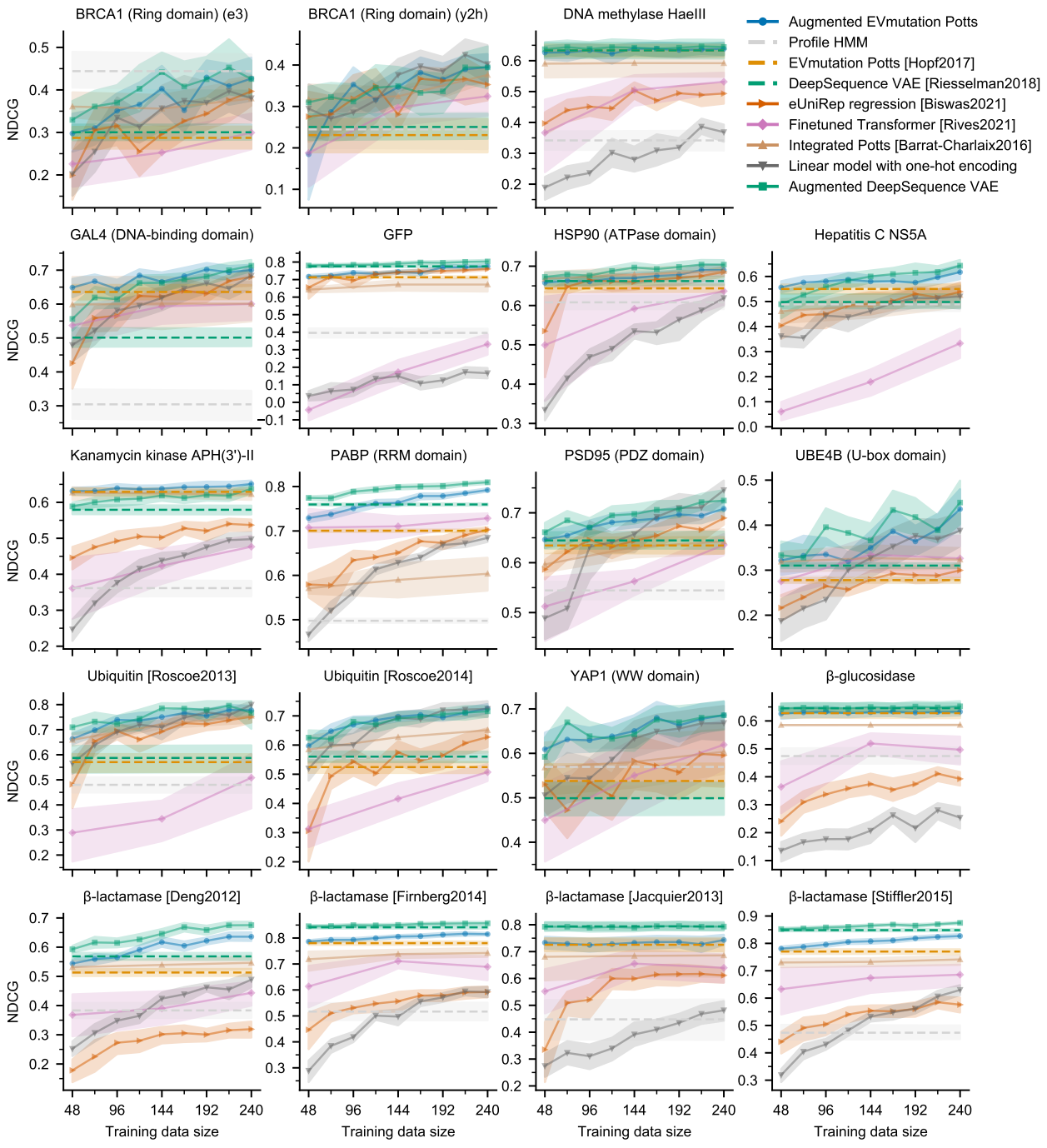

Figure A.1: Performance on individual data sets by NDCG when trained on limited labeled data. Similar to Figure 2.8 but with NDCG instead of Spearman correlations. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

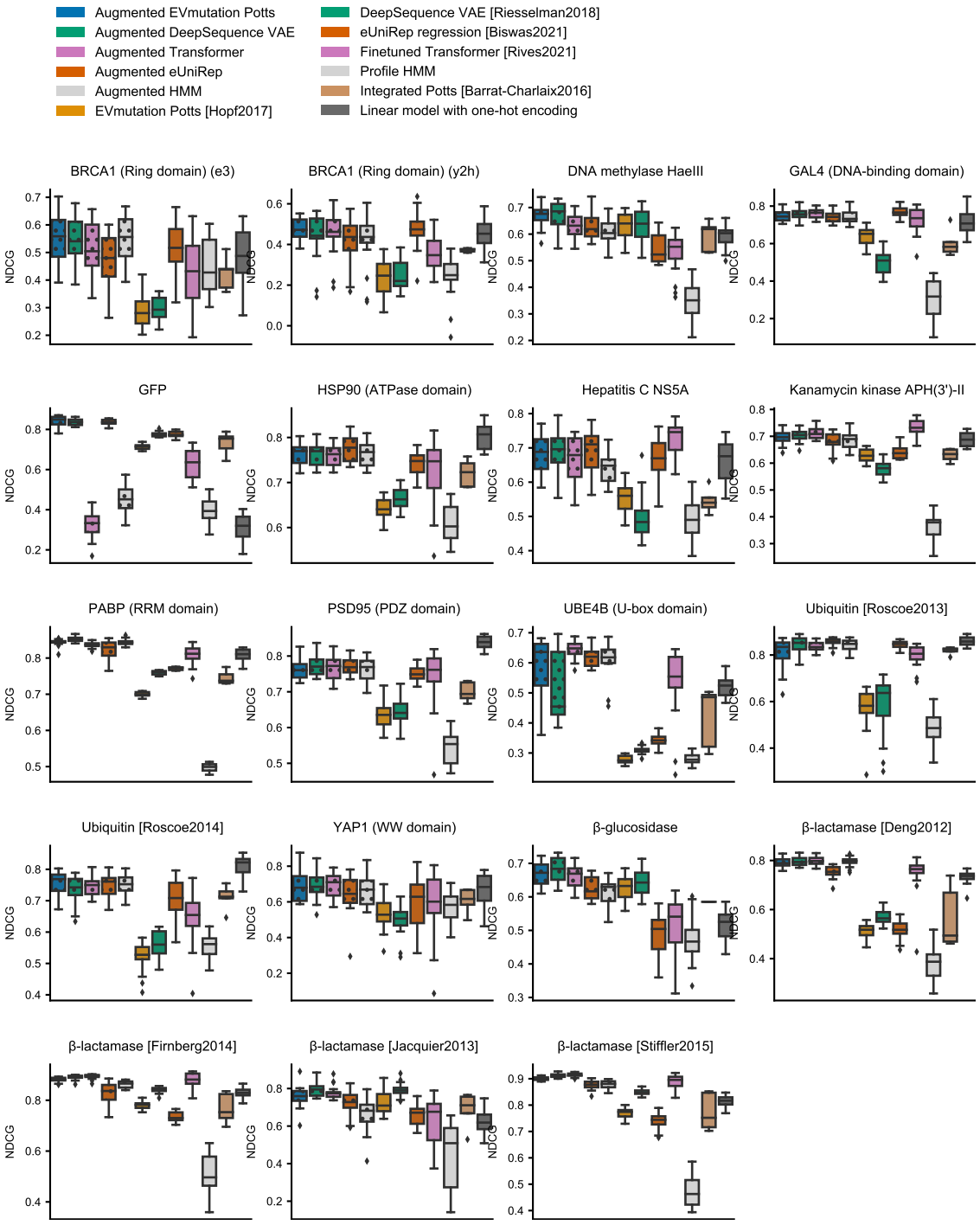

Figure A.2: Performance on individual data sets by NDCG on 80%-20% data splits. Similar to Figure 2.9 but with NDCG instead of Spearman correlations. Error bars indicate bootstrapped 95% confidence interval from 20 random data splits. Box-and-whisker plots show the first and third quartiles as well as median values. The upper and lower whiskers extend from the hinge to the largest or smallest value no further than 1.5× interquartile range from the hinge.

Protein	UniProt ID	Measurement	# Mutations	MSA size	Reference
β-glucosidase	(Custom)	Enzyme activity	2634 (1)	28048	Romero et al., PNAS, 2015
β-lactamase	BLAT_ECOLX	Ampicillin resistance Ampicillin resistance Amoxicillin resistance Stability	4611 (1) 4807 (1) 951 (1) 4808 (1)	8403	Firnberg et al., Mol Biol Evol, 2014 Stiffler et al., Cell, 2015 Jacquier et al., PNAS 2013 Deng et al., JMB, 2012
BRCA 1 (RING domain)	BRCA1_HUMAN	E3 ligase activity BARD1 interaction	1382 (1) 1335 (1)	25828	Starita et al., Genetics, 2015
PSD95 (PDZ domain)	DLG4_RAT	Peptide binding	1578 (1)	102410	McLaughlin et al., Nature, 2012
GAL4 (DNA-binding domain)	GAL4_YEAST	Transcriptional activity	1123 (1)	17521	Kitzmann et al., Nature Methods, 2015
HSP90 (ATPase domain)	HSP82_YEAST	ATPase activity	4104 (1)	15329	Mishra et al., Cell Reports, 2016
Kanamycin kinase APH(3')-II	KKA2_KLEPN	Kinase activity	4385 (1)	12861	Melnikov et al., NAR, 2014
DNA methylase HaeIII	(Custom)	DNA methyltransferase activity	1634 (1)	14115	Rockah-Shmuel et al., PLOS Comp Bio, 2015
Poly(A)-binding protein (RRM domain)	PABP_YEAST	RNA binding 36522 (2)	1188 (1) 36522 (2)	152041	Melamed et al., RNA, 2013
Hepatitis C NS5A	POLG_HCVJF	Viral replication	1632 (1)	8106	Qi et al., PLOS Pathogens, 2014
Ubiquitin	RL401_YEAST	Growth E1 reactivity	1161 (1) 1295 (1)	21448	Roscoe et al, JMB, 2013 Roscoe et al, JMB, 2014
UBE4B (U-box domain)	UBE4B_MOUSE	Ligase activity	613 (1) 21969 (2) 8088 (3) 1404 (4) 216 ($\geq$ 5)	9172	Starita et al., PNAS, 2013
YAP1 (WW domain 1)	YAP1_HUMAN	Peptide binding	319 (1)	40302	Araya et al., PNAS, 2012
Green Fluorescent Protein	(Custom)	Fluorescence	613 (1) 3662 (2) 2026 (3) 844 (4) 630 ($\geq$ 5)	22535	Sarkisyan et al., Nature, 2016

Table A.1: List of all data sets. In the "# Mutations" columns, the numbers in the brackets represent the order of mutants (*i. e.,* the Hamming distance to the wild-type sequence). For example, "12777 (2)" indicates that the data set contains 12777 double-mutant sequences.

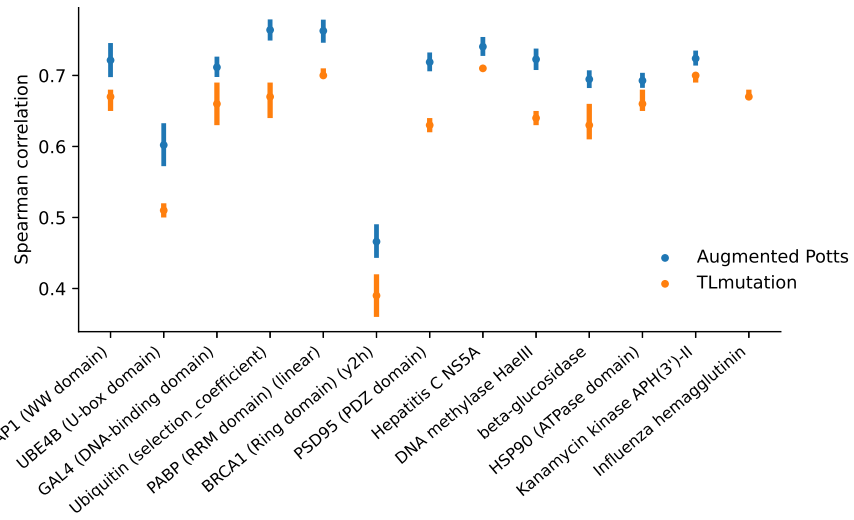

Figure A.3: Comparison of TLmutation and the augmented Potts model. While both TLmutation and augmented Potts models seek to update Potts model parameters with assay-labeled data, augmented Potts models learn additional site-specific parameters to correct the Potts model, whereas TLmutation learns a binary 0/1 mask on the Potts model parameters. When comparing the two methods with random 80-20 train-test splits on the data sets used in TLmutation, the augmented Potts model achieves higher Spearman correlation on most data sets. The TLmutation performance shown is based on Supplementary Figure S1 from Shamsi et al. [114], with error bars indicating 75% confidence interval from five-fold cross-validation. The β-lactamase entry from Supplementary Figure S1 [114] is omitted here as it is unclear which β-lactamase feature column from the data set is referred to.

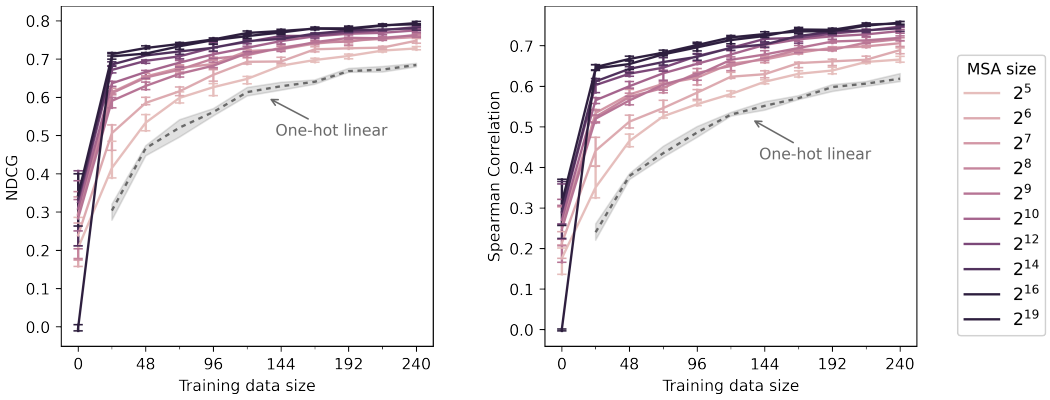

Figure A.4: Effects of MSA size on augmented Potts model performance. With the RRM domain of the poly(A)-binding protein as a case study, we start with the MSA obtained by jackhmmer with bit score 0.5 bits/residue, and then subsequently subsample the MSA randomly. We chose this data set for case study because it is associated with the largest MSA. The performance of the augmented Potts model degrades gradually when subsampling the MSA, but the augmented Potts model still outperforms naive linear regression even with only 32 evolutionary sequences. Error bars are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

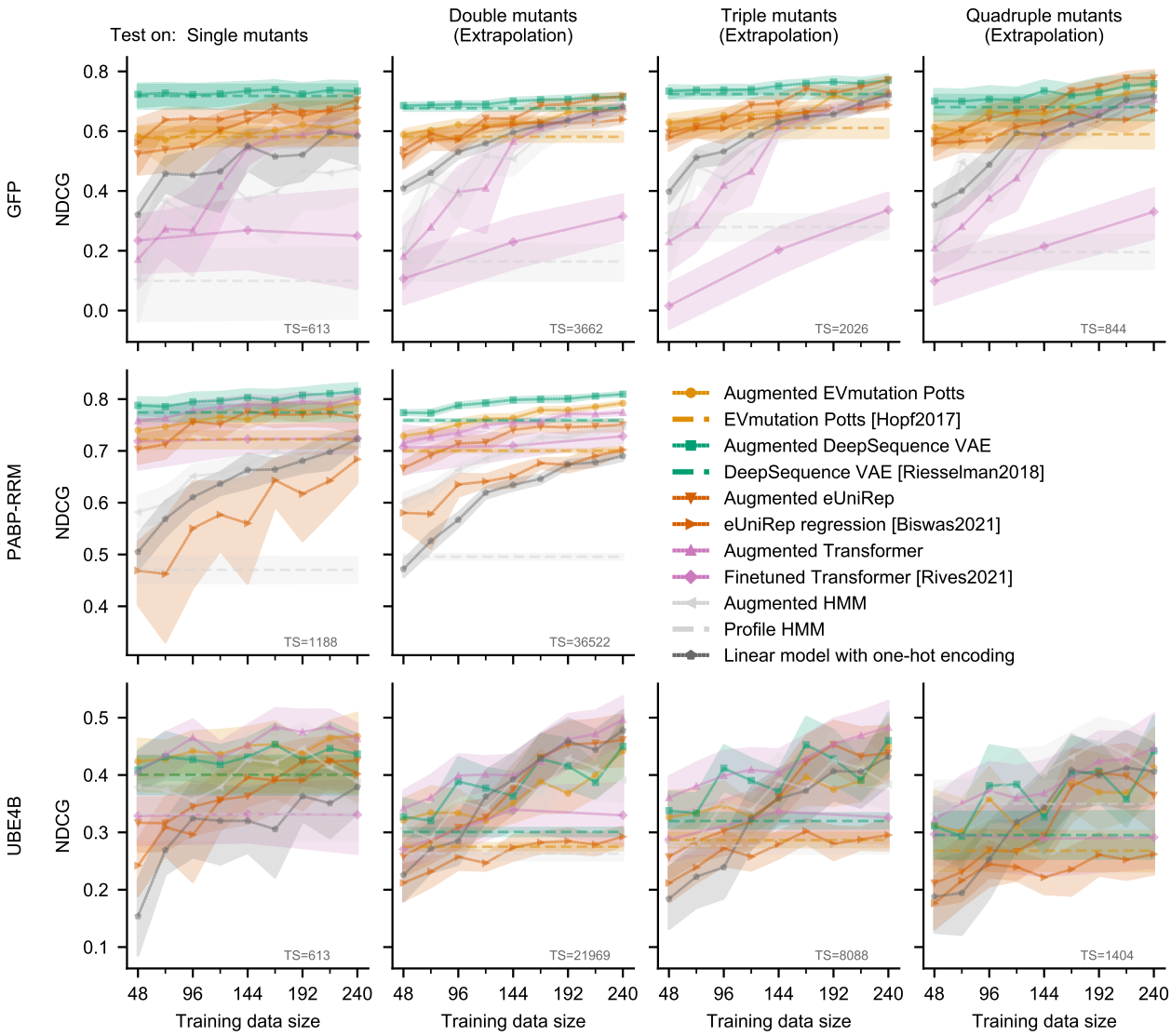

Figure A.5: Extrapolation performance from single mutants to higher-order mutants by NDCG. (NDCG version of Figure 2.5) Here we train on single mutants and test on higher-order mutants on three case studies: the green fluorescent protein (GFP), the RRM domain of Poly(A)-binding protein (PABP), and the U-box domain of ubiquitination factor E4B (UBE4B). Each column measures the ranking quality among mutants that are 1, 2, 3, or 4 mutations away from the wild-type. The "TS" value indicates the total number of mutants of a particular order in all of the data. For example, "TS=613" for single mutants means there were 613 total single mutants in the data set that we sampled from. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

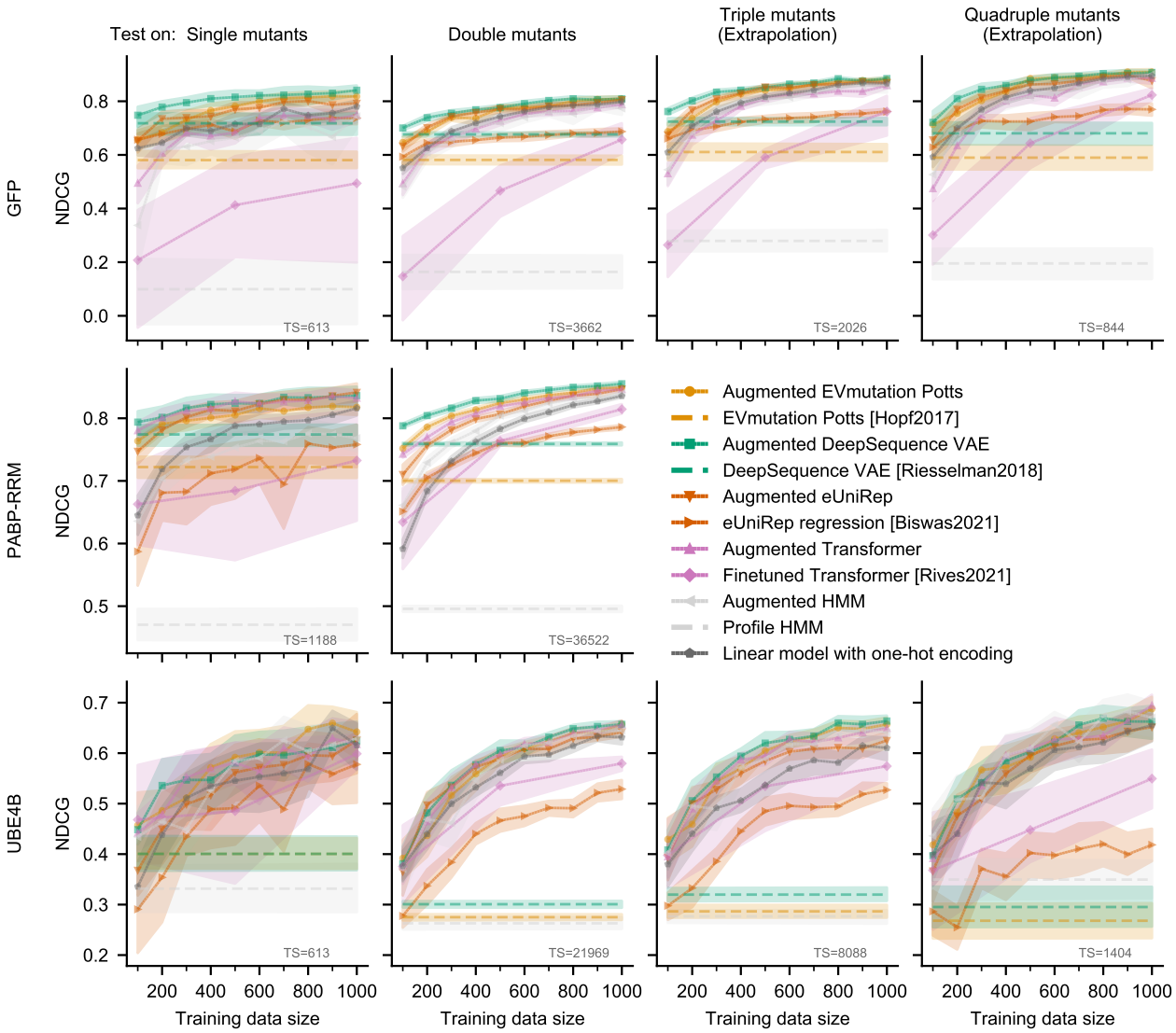

Figure A.6: Extrapolation performance from single and double mutants to higher-order mutants by NDCG. NDCG version of Figure 2.13. Instead of training on single mutants as done in Figure 2.5, here we train on a random sample from both single and double mutants. The "TS" value indicates the total number of mutants of a particular order in all of the data. For example, "TS=613" for single mutants means there were 613 total single mutants in the data set that we sampled from. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

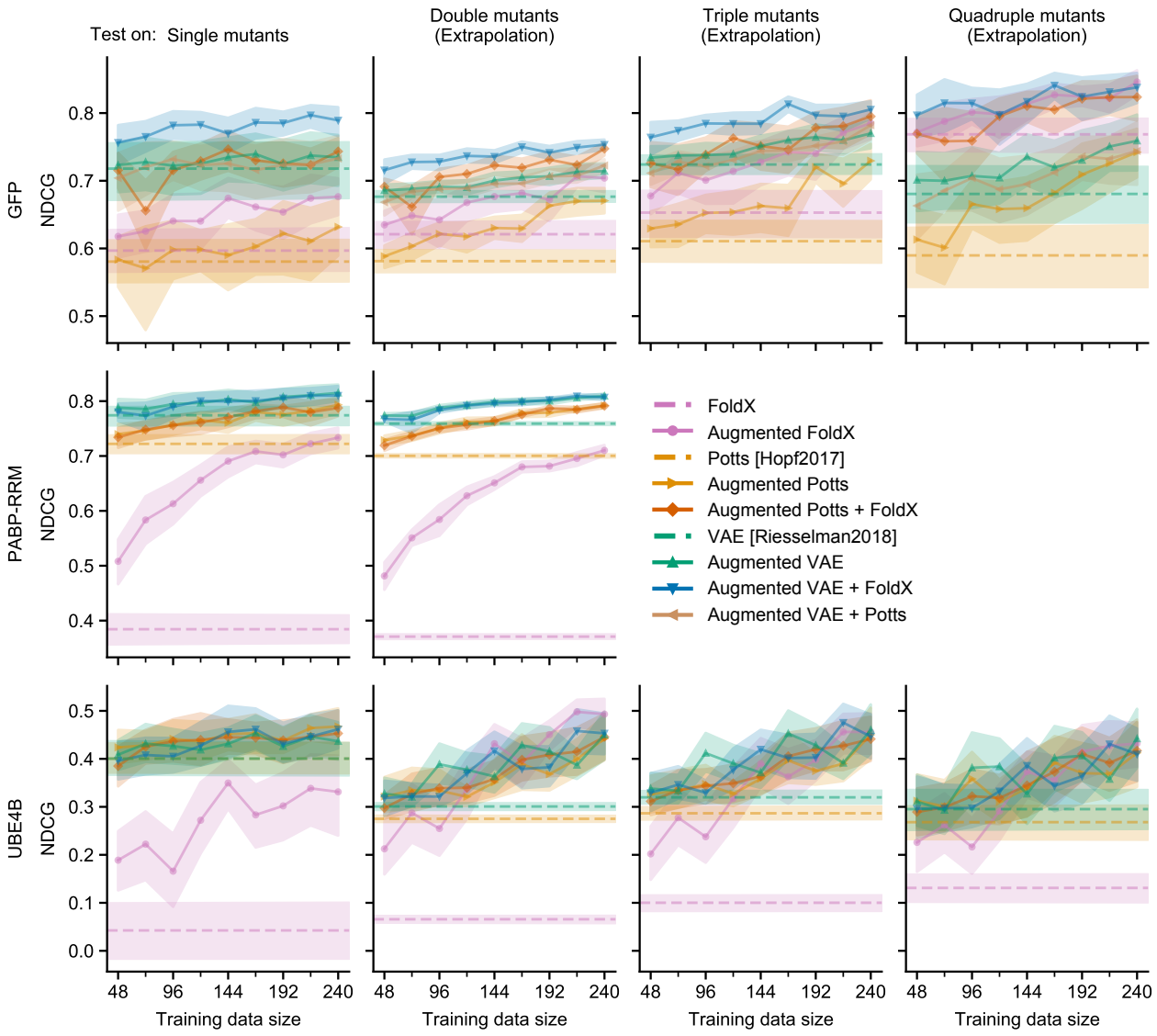

Figure A.7: FoldX predictions as additional features in augmented models, compared by NDCG. NDCG version of Figure 2.15. We evaluate augmented models with FoldX-derived stability features on the three data sets with higher-order mutants. Similar to the Spearman correlation comparison, stability predictions from FoldX as additional features further improves the performance of augmented models as measured by NDCG on the GFP, but not on the poly(A)-binding protein RRM domain and the ubiquitination factor E4B U-box domain. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

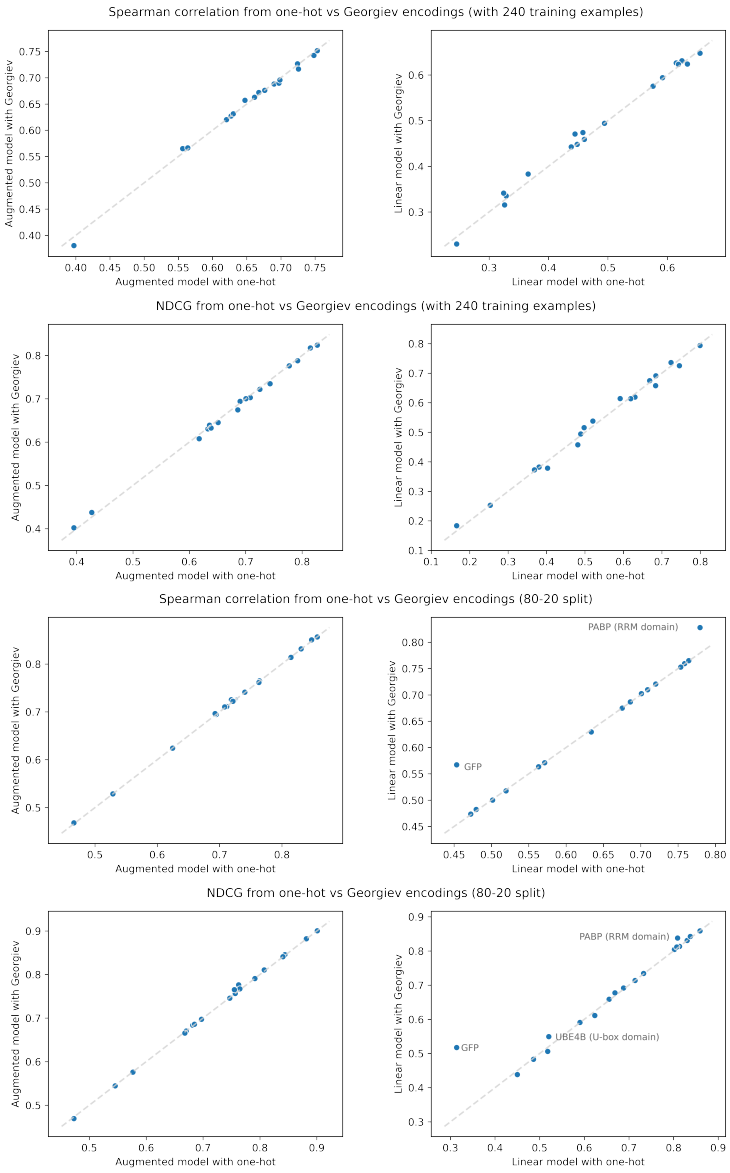

Figure A.8: Comparison of Georgiev encoding and one-hot encoding. We compare models with one-hot encoding to the same models with the 19-dimensional physicochemical representation of the amino acid space developed by Georgiev [44]. Each dot and error bar here represents mean and 95% confidence interval from sampling train and test data with 20 random seeds. On most single-mutant data sets, the Georgiev encoding and the amino acid one-hot encoding result in similar performance. However, on data sets with higher-order mutants when presented with sufficient training data (bottom right), the Georgiev encoding does achieve better performance than the one-hot amino acid encoding, agreeing with previous findings [141]. For the augmented Potts model, the choice of encoding does not influence performance.

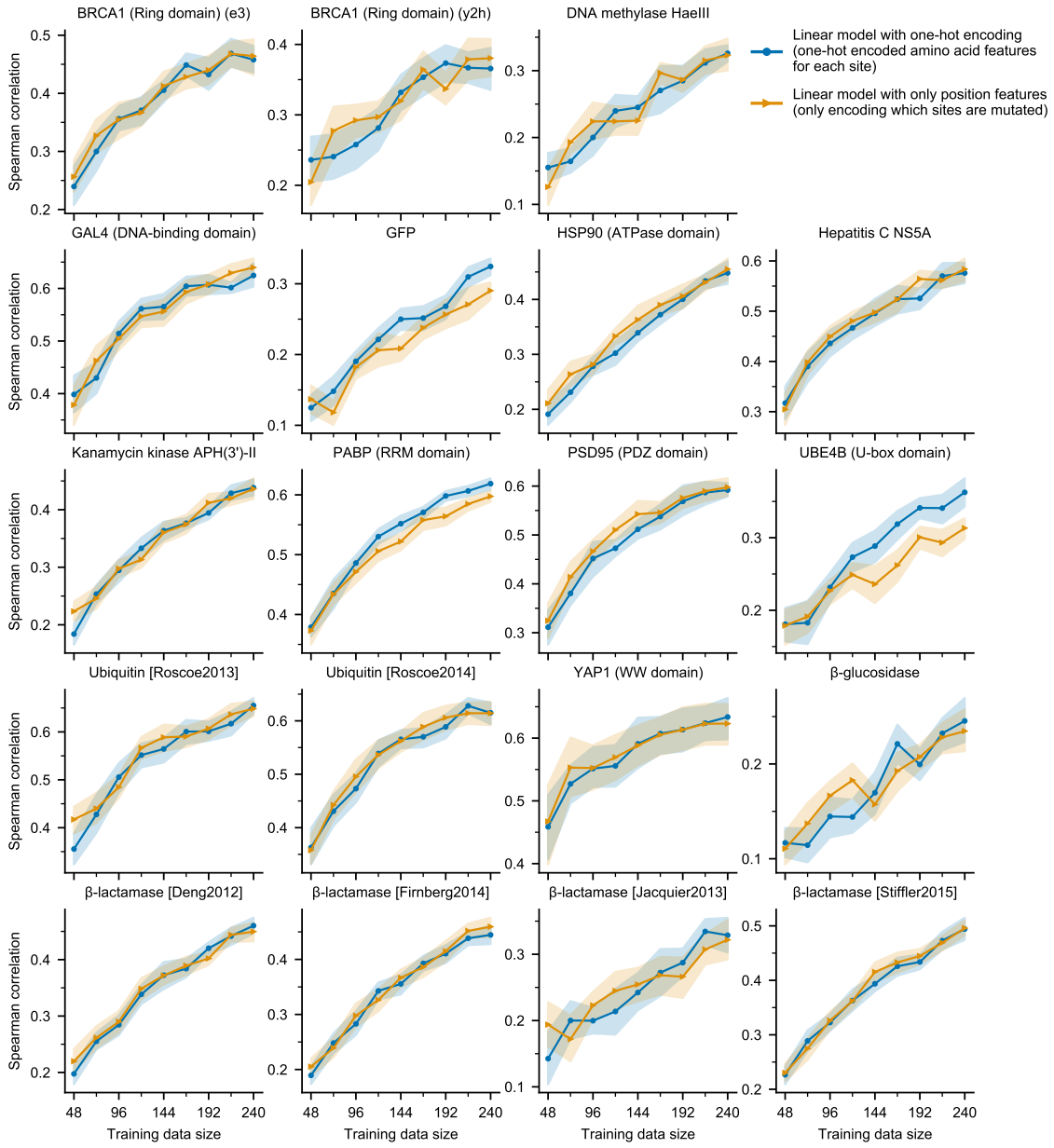

Figure A.9: Performance of linear model with only position features by Spearman correlation. In addition to the linear model with one-hot encoded amino acid features, we also evaluate a simpler linear model with position-only features that encode which sites are mutated. On most data sets, this simpler linear model has comparable performance as the linear model with one-hot encoded amino acid features, despite having much fewer features. On the three data sets with higher-order mutants, however, amino acid features do contribute to higher Spearman correlations. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits. See Figure 2.16 for detailed extrapolative performance on the three data sets with higher-order mutants.

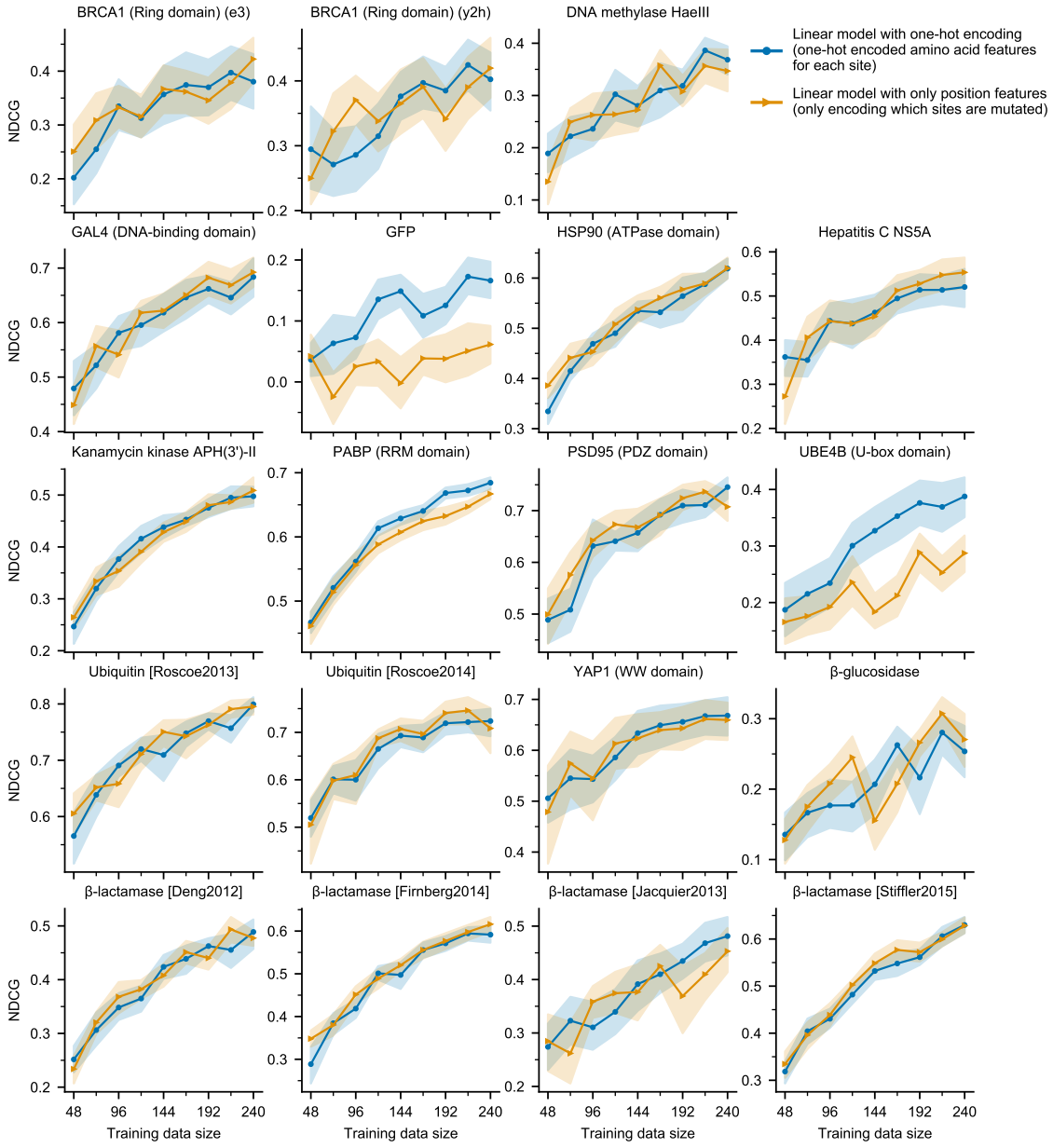

Figure A.10: Performance of linear model with only position features by NDCG. NDCG version of Figure A.9. Similarly to the Spearman correlation result, the simpler linear model with position-only features has comparable performance as the linear model with one-hot encoded amino acid features on most data sets by NDCG, with the exception of the three data sets with higher-order mutants. The gap between the two linear models is more substantial by NDCG than by Spearman on those three datasets. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits. See Figure A.11 for detailed extrapolative performance on the three data sets with higher-order mutants.

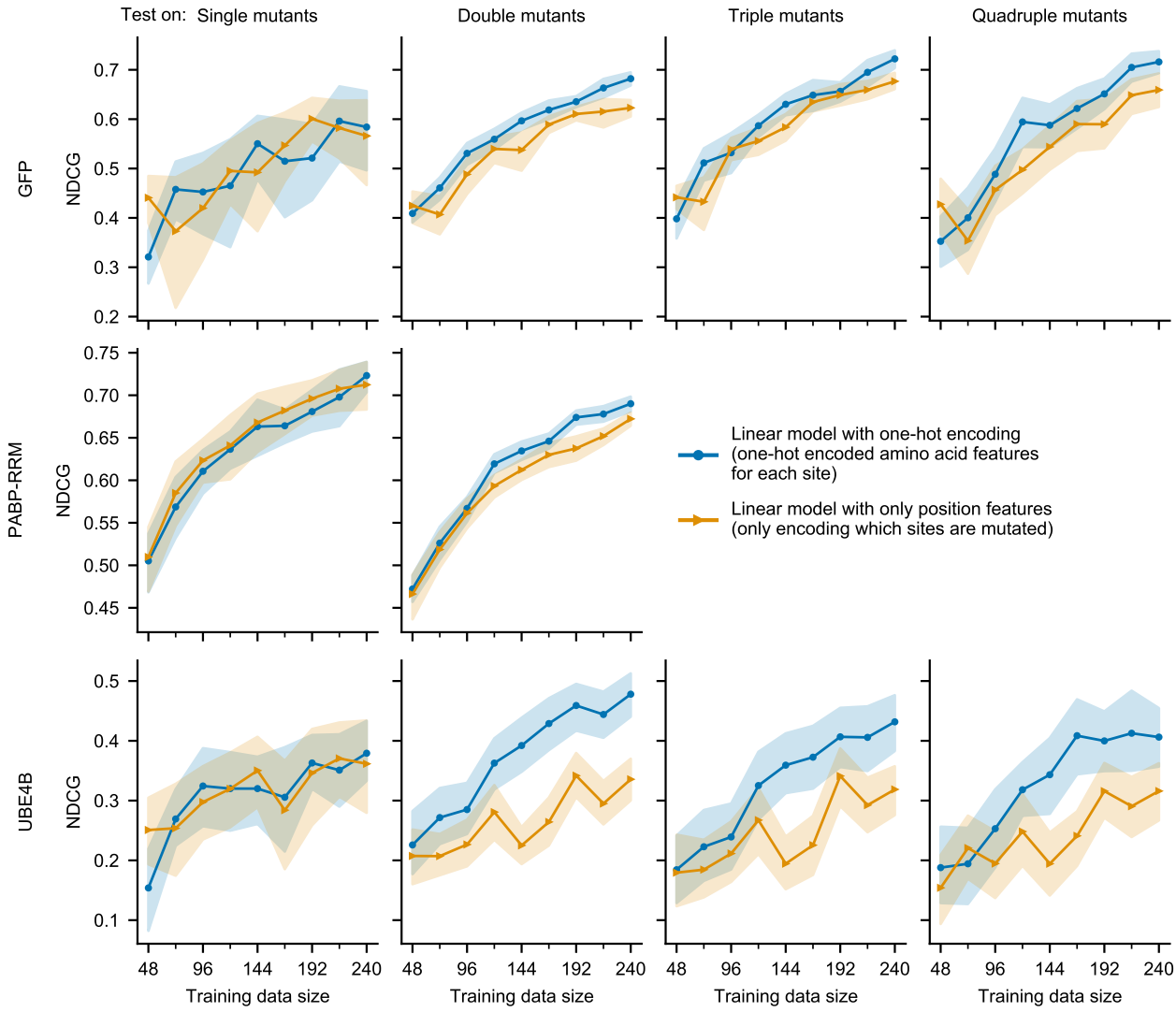

Figure A.11: Extrapolative performance of linear model with only position features by NDCG. NDCG version of Figure 2.16. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits. Similar to the Spearman correlation result, we also see that the linear model with only position-specific features matches the performance of the linear model with one-hot encoded amino acid features on single mutants, but have worse NDCG than the linear model with one-hot encoded amino acid features on double, triple, and quadruple mutants. The performance gap is more substantial by NDCG than by Spearman correlation.

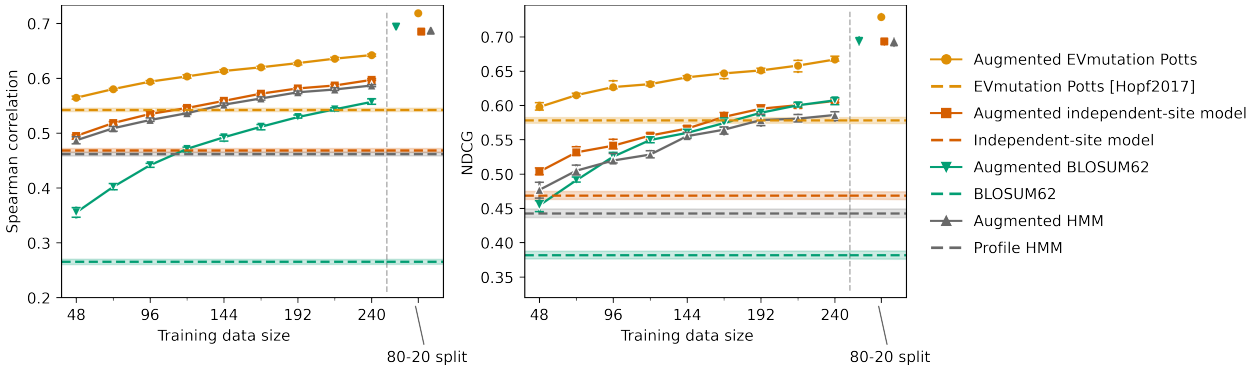

Figure A.12: Comparison of the augmented Potts model and simpler augmented models. With the same setup as Figure 2.3, we compare simpler density models. Error bars are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits. The "independent-site" model refers to the EVmutation [53] independent model implementation, while the "profile HMM" model refers to the HMMER [42] implementation. The difference in optimization algorithms lead to minor performance differences between the two versions. For reference, we also include the augmented Potts model here as comparison.

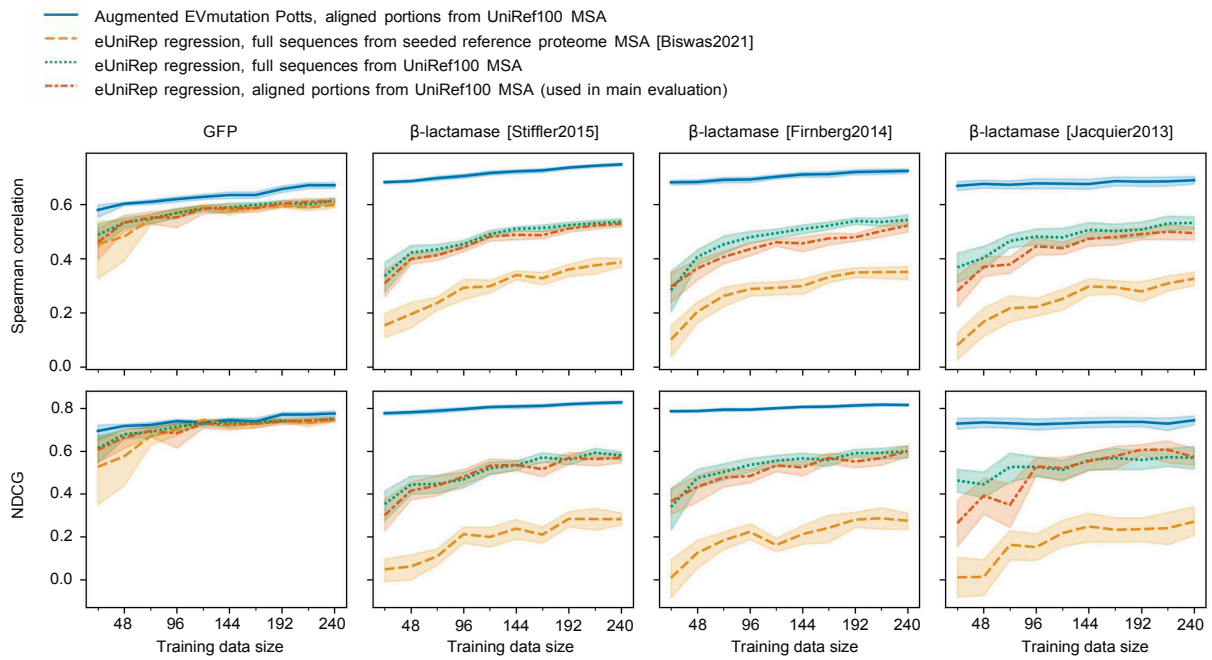

Figure A.13: Comparison of eUniRep evo-tuning procedures. We made two modifications in the evo-tuning procedures for eUniRep models: 1) we used the EVmutation jackhmmer parameters for consistent evolutionary data with other methods, and 2) we focused on aligned portions of sequences rather than unaligned full sequences to lower the computational burden. Here we show these two modifications do not negatively affect performance on GFP and β-lactamase—the two proteins studied by Biswas et al. [17]. The orange and green lines compare the effects of jackhmmer parameters, while the green and red lines compare the effects of using aligned portions instead of full sequences. For GFP, we used the open-sourced eUniRep model [4] as the orange line, while for β-lactamase we used Pfam PF00144 as jackhmmer seed sequences along with default jackhmmer parameters from the webserver and evo-tuned UniRep on the resulting sequences. Error bands are centered at mean and indicate bootstrapped 95% confidence interval from 20 random data splits.

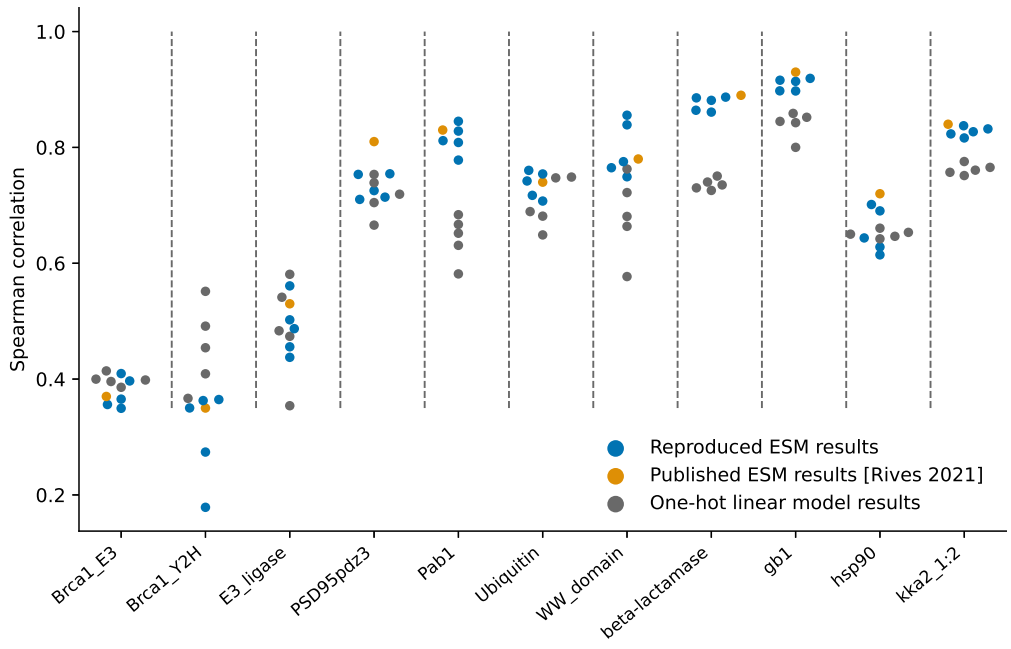

Figure A.14: Reproducing fine-tuned Transformer results on Envision data. When fine-tuning ESM-1b Transformer on the same Envision [46] data as used by Rives et al. [106], we reproduced the Spearman correlation performance for most of the Envision data sets on 80-20 splits. Orange is based on numbers from Supplementary Table S5 by Rives et al. [106], while blue is from our reproduced runs. For each data set, we samples 80% training data with 5 random seeds (20% of the 80% is used for validation to determine early stopping), and report the Spearman correlation on the 20% held-out data.

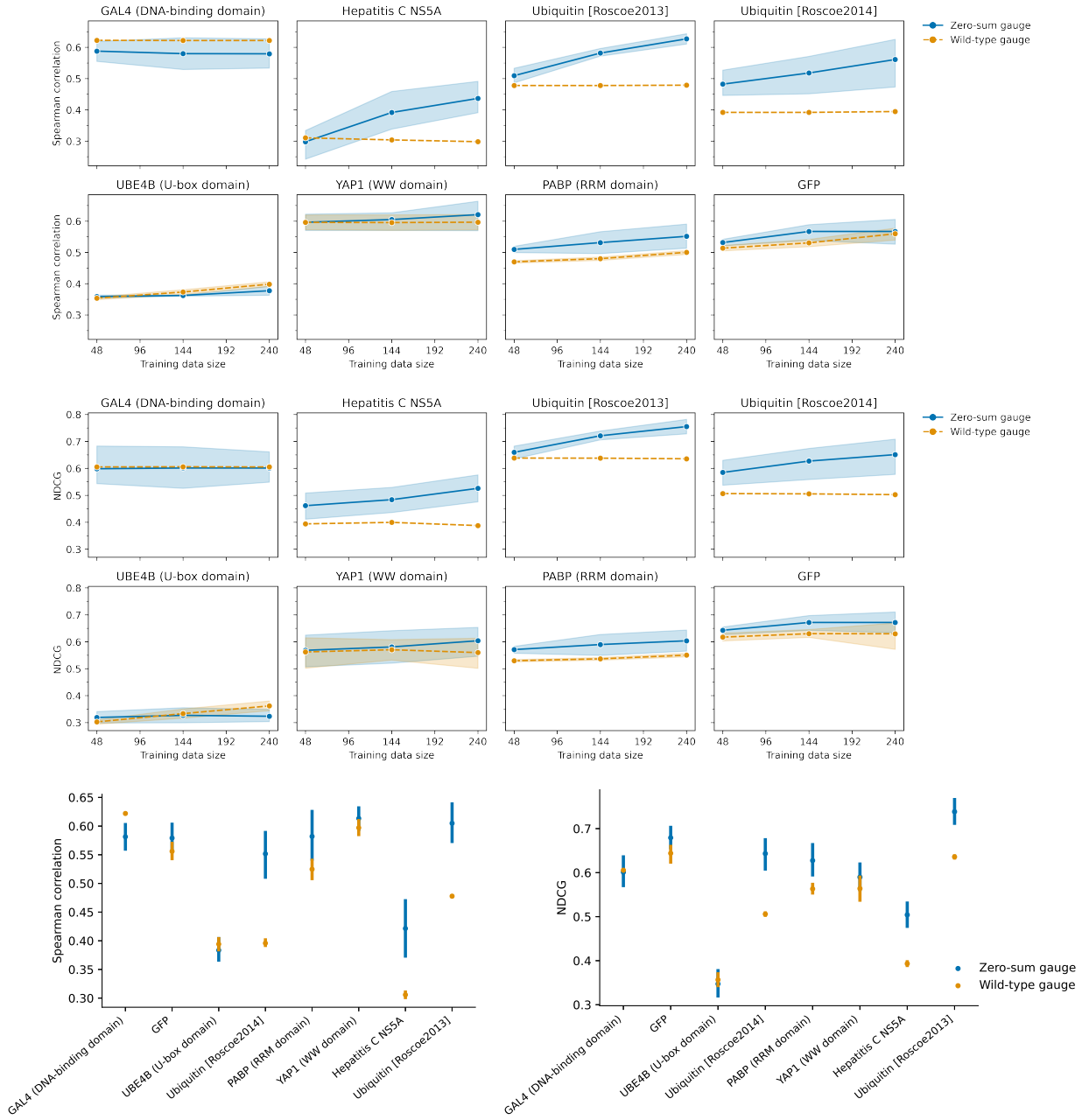

Figure A.15: Comparison of integrated Potts models with wild-type gauge and zero-sum gauge. Top: Spearman correlation comparisons on models trained with 48, 144, 240 training examples. Middle: NDCG comparisons on models trained with 48, 144, 240 training examples. Bottom left: Spearman correlation comparisons on 80-20 splits. Bottom right: NDCG comparisons on 80-20 splits. Overall, zero-sum gauge has achieves higher Spearman correlation than wild-type gauge. Error bands and error bars are centered at mean and indicate bootstrapped 95% confidence interval from 5 random data splits.

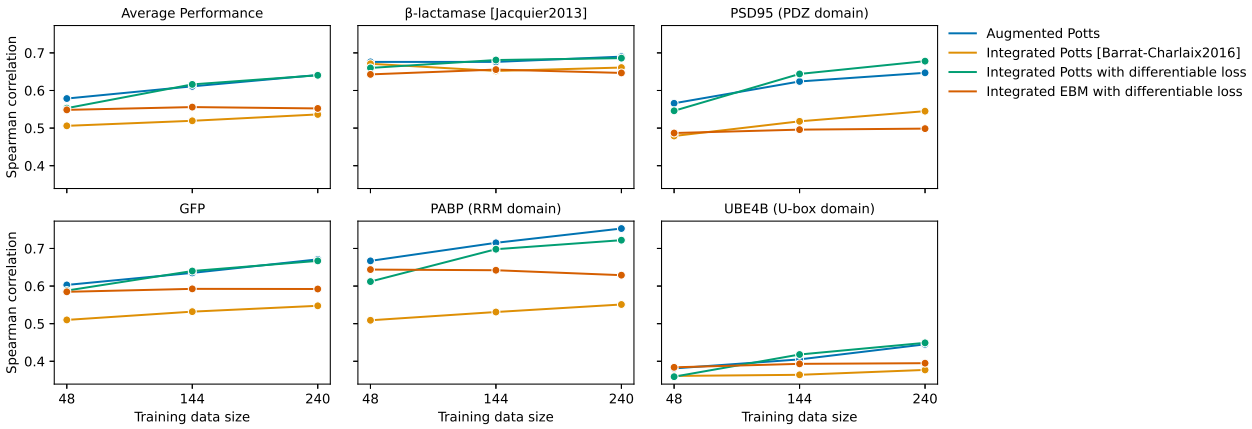

Figure A.16: Comparison of tied-energy models and the augmented Potts model. We evaluated three versions of tied-energy (integrated) models: 1) integrated Potts models [12] with rank-based monotonic mapping [41] between energies and fitness values; 2) integrated Potts models with a differentiable proxy of Spearman correlation as loss; 3) integrated energy-based models with neural network energy function and the same differentiable loss. As integrated models are very expensive to train, we chose to evaluate integrated models on the same β-lactamase and PSD95 data sets as Barrat-Charlaix et al. [12] and also additionally on the three data sets with higher-order mutants. In contrast to tied-energy models where evolutionary data and labeled data are fitted together, the augmented Potts model takes a two step procedure first fitting to evolutionary data and then to labeled data. While the augmented Potts model is orders of magnitude faster to compute, it still results in comparable performance with the best tied-energy model.

Appendix B

Supplementary Information for Chapter 3

B.1 Dataset of Predicted Structures

We used training data from two sources: 1) experimental protein structures from the CATH 40% non-redundant chain set, and 2) AlphaFold2-predicted structures from UniRef50 sequences. To evaluate the generalization performance across different protein folds, we split the train, validation, and test data based on the CATH hierarchical classification of protein structures [94] for both data sources. To achieve that a rigorous structural hold-out, we additionally use foldseek [74] for pairwise TMalign between the test set the train set.

CATH topology split. Following the structural split methodology in previous work [61, 67, 126], we randomly split the CATH v4.3 (latest version) topology classification codes into train, validation, and test sets at a 80/10/10 ratio. The CATH [94] structural hierarchy, classifies domains in four levels: Class (C), Architecture (A), Topology/fold (T), and Homologous superfamily (H). The topology/fold (T) level roughly corresponds to the SCOP fold classification.

Experimental structures. We collected full chains up to length 500 for all domains in the CATH v4.3 40% sequence identity non-redundant set. The experimental structure data contained only stand-alone chains and no multichain complexes. As each chain may be classified with more than one topology codes, we further removed chains with topology codes spanning different splits, so that there is no overlap in topology codes between train, validation, and test. This results in 16,153 chains in the train split, 1457 chains in the validation split, and 1797 chains in the test split.

Predicted structures. We curated a new data set of AlphaFold2 [71]-predicted structures for a selective subset of UniRef50 (202001) sequences. To prevent information leakage about

the test set from the predicted structures, we proceeded in the following steps.

First, we annotated UniRef50 sequences with CATH classification by the Gene3D [79] database, also used by Strokach [126] for data curation. Gene3D represents each CATH classification code as a library of representative profile HMMs. We searched all HMMs associated with the validation and test splits against the UniRef50 sequences using default parameters in hmmsearch [98] and excluded all hits.

Additionally, as AlphaFold2 predictions use multiple sequence alignments (MSAs) as inputs, we also took precaution to avoid information leakage from sequences in the MSAs. We created a filtered version of UniRef100 by searching all the validation-split and test-split Gene3D HMMs against UniRef100 (202001) and excluding all hits. Then, we constructed our MSAs using hhblits [124] on this filtered version of UniRef100. While this filtering step was out of precaution, in retrospect it was perhaps unlikely for the MSA inputs to AlphaFold2 to leak information as the MSAs themselves were not seen during training. The filtering step may have negatively impacted the quality of the resulting predicted structures, although empirically only a very small percentage of MSA sequences were filtered out.

As AlphaFold2 predictions are computationally costly, our budget only allowed for predicting structures for a subset of the UniRef50 sequences. We ranked UniRef50 sequences based on the distogram lDDT score, based on distogram predictions from the MSATransformer model [102], as a proxy for the quality of predicted structures. While the original distogram lDDT score (Supplementary Equation 6 in [112]) is based on pairwise distances from native protein structures, in the absence of native structures we use the argmax of pairwise distances instead, effectively measuring the "sharpness" of distograms and prioritizing sharper predictions. In this order, using AlphaFold2 Model 1 on the filtered UniRef100 MSAs described above, we obtained predicted structures for the top 12 million UniRef50 sequences under length 500, roughly 750 times the CATH train set size.

We used the publicly released model weights from AlphaFold2 Model 1 for CASP14 as a single model, as opposed the 5-model ensemble in [71], to cover more sequences with the same amount of computing resources. We curated the input MSAs from UniRef100 with hhblits, with an additional filtering step as described above. To reduce computational costs, compared to the standard AlphaFold2 protocol, we did not include the UniRef90 jackhmmer MSAs, or the MGnify and BFD metagenomics MSAs, nor the pdb70 templates. Other than a reduced inputs, we followed the default settings in AlphaFold2 open source code, using 3 recycling iterations and the default Amber relaxation protocol. Despite the reduced inputs, the resulting 12 million predicted structures still have high pLDDT scores from AlphaFold, with 75% of residues having pLDDT above 90 (highly confident).

We found that increasing the predicted data size to up to 1 million structures (75 times the CATH experimental data size) substantially improves model performance. Beyond 1 million structures, models still benefit from more data but with diminished marginal returns (Figure 3.6a).

	GVP-GNN	GVP-GNN-large	GVP-Transformer
GVP-GNN embedding dim (node)	(100, 16)	(256, 64)	(1024, 256)
GVP-GNN embedding dim (edge)	(32, 1)	(32, 1)	(32, 1)
Top K neighbors in GVP-GNN	30	30	30
GVP-GNN encoder layers	3	8	4
GVP-GNN decoder layers	3	8	
Transformer embedding dim			512
Feedforward embedding dim			2048
Attention heads			8
Transformer encoder layers			8
Transformer decoder layers			8
Total number of parameters	**1M**	**21M**	**142M**
Batch size (tokens per GPU)	3072	4096	4096
GPUs	1	32	32
CATH:AF2 mixing ratio	1:0	40:1	80:1
Epochs until convergence	84	368	178
Train time per epoch (GPU hours)	0.07	24	88
Total train time (GPU days)	**0.2**	**368**	**653**
Optimizer	Adam	Adam	Adam
Learning rate schedule	Constant	Inverse square root	Inverse square root
Peak learning rate	1.0E-03	1.0E-03	1.0E-03
Initial learning rate		1.0E-07	1.0E-07
Warm-up updates		5000	5000
Gradient clipping	4.0	1	

Table B.1: Details on model hyperparameters and training.

Noise on AlphaFold2-predicted backbone coordinates. Even after Amber relaxation, the backbone coordinates predicted by AlphaFold2 contain artifacts in the sub-Angstrom scale that may give away amino acid identities. Without adding noise on predicted structures, there is a substantial gap between held-out set performance on predicted structures and on experimental structures. To prevent the model from learning non-generalizable AlphaFold2-specific rules, we added Gaussian noise at the 0.1A scale on predicted backbone coordinates. The Gaussian noise improves the invariant Transformer performance but not the GVP-GNN performance (Supplementary Figure B.3).

B.2 Span Masking

We add a binary feature indicating whether each coordinate is masked or not. In GVP-Transformer, we exclude the masked nodes in the GVP-GNN encoder layers, and then impute zeros when passing the GVP-GNN outputs into the main Transformer. Imputing zeros for missing vector features ensure the rotation- and translation- invariance of the model. In GVP-GNN, we impute zeros for the input vector features, and in the input graph connect the masked nodes to the k sequence nearest-neighbors ($k = 30$) in lieu of the k nearest nodes by spatial distance.

For span masking, we randomly select continuous spans of up to 30 amino acids until 15% of input backbone coordinates are masked. Such a span masking scheme has shown to improve performance on natural language processing benchmarks [70]. The span lengths are sampled from a geometric distribution $\text{Geo}(p)$ where $p = 0.05$ (corresponding to an average span length of $1/p = 20$). The starting points for the spans are uniformly randomly sampled. Compared to independent random masking, span masking is better for GVP-Transformer but not for GVP-GNN (Table B.3).

For the amino acids with masked coordinates, we exclude the corresponding nodes from the input graph to the pre-processing GVP message passing layers, and then impute zeros for the geometric features when passing the GVP outputs into the main Transformer. Imputing zeros for missing vector features ensure the rotation- and translation- invariance of the model.

B.3 Model Architectures

Autoregressive modeling. GVP-GNN and GVP-Transformer both have encoder-decoder architectures. The encoder only receives the structural features. The decoder receives the encoder output along with the one-hot encoding of the amino acids. In the autoregresive decoder, sequence information only propagate from amino acid i to j for $i < j$. The last decoder layer produces a 20-way scalar output per position and softmax activation to predict the probabilities for the amino acid identity at the next position in the sequence.

Invariance to rotation and translation. The input features for both GVP-GNN and GVP-Transformer are translation-invariant, making the overall models also invariant to translations.

Each GVP-GNN layer is rotation-equivariant, that is, for a vector feature x and any arbitrary rotation T, $Tf(x) = f(Tx)$. With equivariant intermediate layers and an invariant output projection layer, GVP-GNN is overall invariant to rotations, since the composition of an equivariant function f with an invariant function g produces an invariant function $g(f(x))$.

The GVP-Transformer architecture is also invariant to rotations and translations. The initial GVP-GNN layers in GVP-Transformer output rotation-invariant scalar features and rotation-equivariant vector features for each amino acid. To make the overall GVP-Transformer invariant, we perform a change of basis on GVP-GNN vector outputs to produce rotation-

		Perplexity			Recovery %		
Model	Data	Short	Single-chain	All	Short	Single-chain	All
Structured GNN	CATH	10.08	7.04	7.06	27.8%	35.1%	35.4%
GVP-GNN	CATH	8.13	5.76	5.86	31.5%	41.1%	40.4%
	+ AlphaFold2	9.87	6.61	6.50	26.3%	36.3%.	36.8%
GVP-GNN-large	CATH	8.87	6.62	6.68	31.0%	37.2%	37.4%
	+ AlphaFold2	7.08	4.46	4.39	**34.1%**	48.2%	48.7%
GVP-Transformer	CATH	8.80	6.78	6.97	28.5%	36.7%	36.3%
	+ AlphaFold2	**6.99**	**4.36**	**4.34**	33.0%	**48.9%**	**49.5%**

Table B.2: Fixed backbone sequence design performance on the more stringent structurally held-out test set from CATH v4.3 chains (and its short and single-chain subsets) in terms of per-residue perplexity (lower is better) and recovery (higher is better).

invariant features for the Transformer. More specifically, for each amino acid, we define a local reference frame based on the N, CA, and C atom positions in the amino acid, following Algorithm 21 in AlphaFold2 [71]. We then perform a change of basis according to this local reference frame, rotating the vector features in GVP-GNN outputs into the local reference frames of each amino acid. (If GVP-GNN outputs are used directly as Transformer inputs without this change of basis, the GVP-Transformer model would not be rotation-invariant.) We concatenate this rotated "local version" of vector features together with the scalar features as inputs to the Transformer. The concatenated features are invariant to both translations and rotations on the input backbone coordinates, forming a $L \times E$ matrix where L is the number of amino acids in the protein backbone and E is the feature dimension. For amino acids with masked or missing coordinates, the features are imputed as zeros.

Transformer. We closely followed the original autoregressive encoder-decoder Transformer architecture [134] except for using learned positional embeddings instead of sinusoidal positional embeddings, attention dropout, and layer normalization inside the residual blocks ("pre-layernorm"). For model scaling experiments, we followed the model sizes in [132], and chose the 142-million-parameter model with 8 encoder layers, 8 decoder layers, 8 attention heads, and embedding dimension 512 based on the best validation set performance (Figure 3.6c shows test set ablation).

The GVP-GNN, GVP-GNN-large, and GVP-Transformer models used in the evaluations in this manuscript are all trained to convergence, with detailed hyperparameters listed in Table B.1.

B.4 TM-score-based Test Set

In addition to the CATH topology-based test set following previous work [61, 67], we also create an even more stringent test set based on pairwise TM-score comparison between train and test examples. The CATH topology split does not completely prevent high TM-score

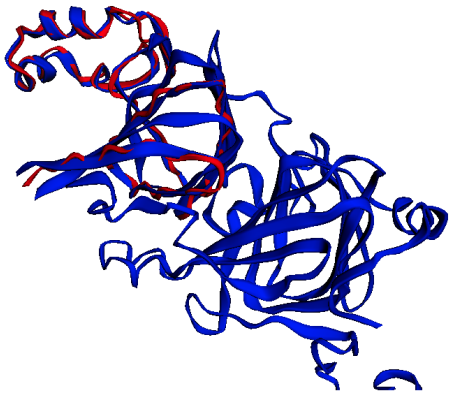

Figure B.1: An illustrative example of structural overlap between CATH topology splits. The jack bean canavalin (PDB code 1DGW; chain Y; red) and the soybean β-Conglycinin (PDB code 1UIJ; chain B; blue) are assigned different topology codes in CATH (1.10.10 and 2.60.120), but they align with TM-score 0.94 and CA RMSD 0.7A on a segment of 90 residues. The difference in topology classifications likely resulted from CATH annotating only a 37-residue mainly helical segment of the jack bean canavalin as a domain while annotating a longer 176-residue mainly beta sheet segment of the soybean β-Conglycinin as a domain.

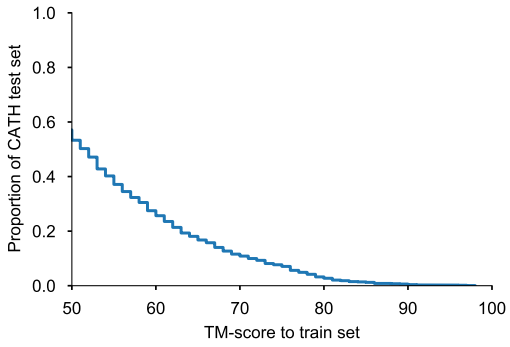

Figure B.2: Distribution of the highest TM-score from each test example to the train set. For example, 54% of the CATH topology split test set has at least one match in the train set with TM-score above 0.5, and 27% of the topology split test set has at least one match in the train set with TM-score above 0.6.

matches between train and test structures. We illustrate such an example in Figure B.1, and show overall TM-score statistics in Figure B.2.

We constructed a TM-score-based test set of 223 proteins with no TMalign matches (TM-score ≥ 0.5) from the train set, using the foldseek [74] TMalign tool with default parameters for the pairwise search.

We found that the conclusions about model performance overall remains the same on this TM-score-based test set as on the CATH topology split test set. For consistency with prior work, we report metrics on the CATH topology test set in the main manuscript, while showing metrics on the smaller TM-score-based test set in Table B.2.

B.5 Additional Results and Details

Ablation on noise and masking during training. We found that GVP-Transformer models trained with Gaussian noise during training perform slightly better at test time than those trained without (Table B.3). When given full backbone coordinates at test time, training with span masking only very slightly improves model performance compared to no masking or to random masking, even though there is a much larger performance gap between random masking and span masking on regions with masked backbone coordinates (Figure 3.4).

Dual-state design test set from PDBFlex. We test design performance on multiple conformations by finding test split proteins with distinct conformations in the PDBFlex database. From PDBFlex, we looks for experimental structures of protein sequences in the CATH topology split test set (95% sequence identity or above), and take all paired instances that are at least 5 angstroms apart in overall RMSD between conformations. We report perplexity on locally flexible residues (defined as local RMSD above 1 angstrom). To be more conservative in our evaluation, we show the better of the two conformations to represent single-state perplexity in Figure 3.7.

Ablation on the number of GVP-GNN encoder layers in GVP-Transformer. Increasing the number of GVP-GNN encoder layers improves the overall model performance (Figure B.3), indicating that the geometric reasoning capability in GVP-GNN is complementary to the Transformer layers.

Model performance when trained only on predicted structures. When trained on the 12 million predicted structures without including any of the experimental structures from CATH in training data, the model performance of GVP-GNN, GVP-GNN-large, and GVP-Transformer is across the board substantially worse than when trained only on the CATH structures (Table B.4). This gap is especially pronounced for the larger GVP-GNN-large and GVP-Transformer models.

			Perplexity
GVP-Transformer	Span masking	Gaussian noise	4.10
	Span masking	No noise	4.32
	Independent random masking	Gaussian noise	4.30
	No masking	Gaussian noise	4.20

Table B.3: Effects of adding Gaussian noise to predicted structures and effects of span masking during training, as measured by perplexity on CATH topology split test set. The GVP-Transformer has 142M parameters and is trained with a mixing ratio of 1:40.

	Perplexity
GVP-GNN	6.52
GVP-GNN-large	11.51
GVP-Transformer	10.95

Table B.4: Model performance when trained only using the 12 million predicted structures without CATH training data, as measured by perplexity on CATH topology split test set.

	Pearson correlation			
Fold	Structured GNN [61]	GVP-GNN [66]	GVP-GNN-large+AF2	GVP-Transformer+AF2
$\beta\beta\alpha\beta\beta_{37}$	0.47	0.53	0.62	**0.70**
$\beta\beta\alpha\beta\beta_{1498}$	**0.45**	0.39	0.37	0.33
$\beta\beta\alpha\beta\beta_{1702}$	0.12	**0.26**	0.24	0.22
$\beta\beta\alpha\beta\beta_{1716}$	0.47	0.57	**0.60**	0.58
$\alpha\beta\beta\alpha_{779}$	0.57	0.48	0.62	**0.64**
$\alpha\beta\beta\alpha_{223}$	0.36	0.47	**0.57**	0.55
$\alpha\beta\beta\alpha_{726}$	0.21	0.19	0.24	**0.26**
$\alpha\beta\beta\alpha_{872}$	0.23	0.39	0.38	**0.42**
$\alpha\alpha\alpha_{134}$	0.36	0.44	0.46	**0.50**
$\alpha\alpha\alpha_{138}$	0.41	0.44	0.55	**0.58**
Average	0.37	0.42	0.47	**0.48**

Table B.5: Mutation stability prediction performance for small *de novo* proteins [107], with highest correlation bolded.

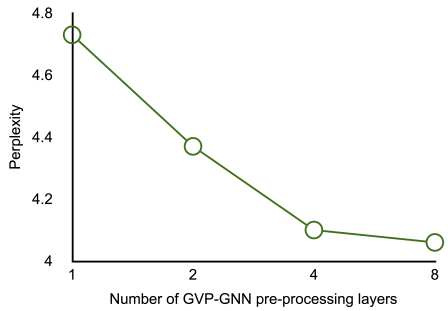

Figure B.3: Effects of varying the number of GVP-GNN pre-processing layers in the GVP-Transformer model, as measured by perplexity on CATH topology split test set.

Stability prediction on *de novo* small proteins. We predict protein stability on an experimentally measured stability dataset for *de novo* small proteins [107]. We use the relative difference in sequence conditional log-likelihoods as a predictor for stability and compute Pearson correlation with the mutation effect following [61], assuming that more stable sequences should score higher in log-likelihoods. For each fold, Rocklin et al. [107] starts with a reference protein and generates sequence variants with single amino acid substitutions. We calculate the Pearson correlation between sequence conditional log-likelihood scores and experimental stability measurements for all designed sequences in each fold. With predicted structures as additional training data, the GVP-Transformer model improves the pearson correlation on 8 out of the 10 folds.

Perplexity and sequence recovery of SARS-CoV-2 RBD. We show perplexity and sequence recovery on the SARS-CoV-2 protein receptor binding domain (RBD) as an example for inverse folding. The RBD can exist in a closed-state with the RBD down or in an open-state with the RBD up [136], as illustrated in Figure B.5. The SARS-Cov-2 spike protein structure has no match with the training data with TM-score above 0.5. The SARS-Cov-2 spike protein has both an open and closed state (open state: PDB 6XRA; closed state: PDB 6VXX). We evaluate perplexity and sequence recovery conditioning on each of the two states independently and jointly. Conditioning on the open state results in better perplexity and sequence recovery than conditioning on the closed state. Conditioning on both states gives improvement in both perplexity and sequence recovery compared to conditioning only on the open state.

	Perplexity			Recovery %		
	Open state	Closed state	Dual-state	Open-state	Closed state	Dual-state
GVP-GNN	4.64	5.13	4.20	45.3%	44.2%	49.7%
GVP-Transformer	4.50	4.96	4.06	49.2%	48.1%	53.6%

Table B.6: Perplexity and sequence recovery on the SARS-Cov-2 spike protein receptor binding domain (RBD), conditioned on either the closed state, the open state, or both states (illustrated in Figure B.5). The inputs to inverse folding models consist of the backbone coordinates for the entire spike protein, while the perplexity evaluation is only on the RBD.

		AUROC
Supervised	3DCNN	0.57
	GNN	0.62
	ENN	0.57
	GVP-GNN	0.68
Transfer	GVP-GNN	**0.71**
Zero-shot	GVP-GNN (chain)	0.58
	GVP-GNN (complex)	**0.71**
	GVP-GNN-large+AF2 (chain)	0.61
	GVP-GNN-large+AF2 (complex)	**0.71**
	GVP-Transformer+AF2 (chain)	0.60
	GVP-Transformer+AF2 (complex)	0.68

Table B.7: Protein complex stability on SKEMPI test set (binary classification of increase in stability on single-point mutations). Although only trained on single chains, the inverse-folding models generalize to protein complexes. Giving the full complex as input, *complex*, improves performance compared to giving only the chain as input, *chain*. Zero-shot prediction compared to fully supervised and supervised transfer learning methods from [129] and [66] trained on the SKEMPI train set.

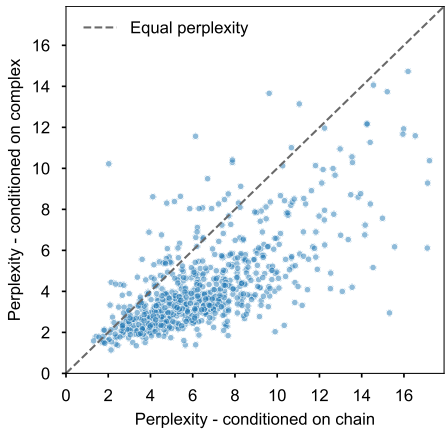

Figure B.4: Fixed backbone sequence design perplexity for protein complexes. The model is evaluated on 796 structurally held-out protein complexes. Comparison of conditioning on the backbone coordinates of individual chains (x-axis) with conditioning on backbone coordinates of the entire complex (y-axis). Note that for both values perplexity is evaluated on the same chain in the complex. The shift to the lower right indicates improved perplexity when the model is given the complete structure of the complex.

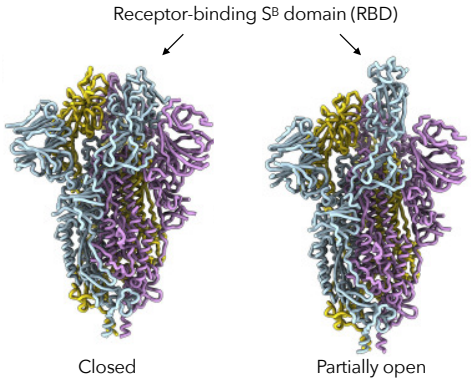

Figure B.5: Illustration of the closed and open states of the SARS-CoV-2 spike protein receptor-binding domain. Cryo-EM structures from [136] (open state: PDB 6XRA; closed state: PDB 6VXX).

	Spearman correlation (zero-shot)			
Evaluation subset	ESM-1v	GVP-GNN	GVP-GNN-large+AF2	GVP-Transformer+AF2
Mutated	-0.23 ± 0.03	$\mathbf{0.34 \pm 0.02}$	0.29 ± 0.03	$\mathbf{0.31 \pm 0.03}$
Designed	0.42 ± 0.02	0.65 ± 0.01	$\mathbf{0.72 \pm 0.01}$	0.67 ± 0.02
High-fitness	0.22 ± 0.02	0.13 ± 0.03	0.21 ± 0.03	$\mathbf{0.26 \pm 0.02}$
Sampled	-0.21 ± 0.03	$\mathbf{0.35 \pm 0.02}$	0.30 ± 0.02	0.30 ± 0.03
≥ 2 mutations	-0.20 ± 0.03	$\mathbf{0.35 \pm 0.03}$	0.29 ± 0.04	0.30 ± 0.02
≥ 3 mutations	0.28 ± 0.03	0.53 ± 0.02	$\mathbf{0.62 \pm 0.02}$	$\mathbf{0.64 \pm 0.02}$
≥ 8 mutations	0.20 ± 0.03	0.47 ± 0.02	$\mathbf{0.53 \pm 0.02}$	$\mathbf{0.55 \pm 0.02}$

Table B.8: Zero-shot performance on AAV split [27].

Predicting RBD-ACE2 binding affinity. We used the binding affinity dataset provided by Starr et al. [122] (https://github.com/jbloomlab/SARS-CoV-2-RBD_DMS), restricting to sites within the RBM subsequence. We used the RBD-ACE2 structure determined by Lan et al. [76] (PDB: 6M0J). For mutational effect predictions with ESM-1v, ESM-1b, and ESM-MSA-1b, we scored mutations using the masked-marginal likelihood ratio between the mutant and wildtype amino acids. To generate the MSA used as input to ESM-MSA-1b, we searched `uniclust30_2017_07` [90] with `hhblits` [124] (using two iterations and an E-value cutoff of 0.001) based on the RBD wildtype sequence as the query.

Predicting complex stability changes upon mutations. SKEMPI [63] is a database of binding free energy changes upon single point mutations within protein complex interfaces. This database is used as a task in the Atom3D benchmark suite [129] for comparing supervised stability prediction methods. The task is to classify whether the stability of the complex increases as a result of the mutation. We compare zero-shot predictions using inverse folding models to supervised and transfer learning methods [129, 66] on the Atom3D test set. We find that sequence log-likelihoods from GVP-GNN, GVP-GNN-large, and GVP-Transformer models are all effective zero-shot predictors of stability changes of protein complexes (Table B.7), performing comparably to the best supervised method which uses transfer learning.

Predicting insertion effects on AAV. Using masked coordinate tokens at insertion regions, inverse folding models can also predict the effects of sequence insertions. Adeno-associated virus (AAV) capsids are a promising gene delivery vehicle, approved by the US Food and Drug Administration for use as gene delivery vectors in humans. Focusing on mutating a 28-amino acid segment, Bryant et al. [21] generated more than 200,000 variants of AAV sequences with 12–29 mutations across this region, and measured their ability to package of a DNA payload. This dataset is unique compared to many other mutagenesis datasets in that most sequences feature random insertions in the 28-amino acid segment, as opposed to only random substitutions.

We use inverse folding models to predict insertion and substitution effects as follows: For each sequence, we input the full backbone coordinates of the wild-type (PDB: 1LP3), and insert one masked token into the input backbone coordinates for each insertion. Then we compare the conditional sequence log-likelihood on this input with masks to the conditional sequence log-likelihood of the wild-type sequence on the wild-type backbone. The difference in these two conditional log-likelihoods are used as the score for predicting packaging ability.

We report the zero-shot performance on each of the 7 data subsets evaluated in the FLIP [27] benchmark suite. For amino acid insertions (marked as lowercase letters in the FLIP data), the corresponding backbone coordinates for those amino acids are marked as unknown in the input structure. As shown in Table B.8, GVP-Transformer trained with predicted structures outperforms the sequence-only zero-shot prediction baseline ESM-1v on

6 out of the 7 data subsets. The reported standard deviations are calculated by sampling different subsets of 10,000 variants from the evaluation data.

For ESM-1v, we scored variant sequences based on the independent marginals formula as described in Equation 1 from Meier et al. [88], scoring mutations using the log odds ratio at the mutated position, assuming an additive model when a set of multiple mutations T exist in the same sequence:

$$\sum_{t \in T} \log p(x_t = x_t^{mt} | x_{\setminus T}^{ins}) - \log p(x_t = x_t^{wt} | x_{\setminus T}) \tag{B.1}$$

where $x_{\setminus T}^{ins}$ in the first term is the wild-type sequence with mask tokens at insertion positions and $x_{\setminus T}$ in the second term is the wild-type sequence without insertions.

Confusion matrix. We calculated the substitution scores between native sequences and sampled sequences (sampled with temperature $T = 1$) by using the same log odds ratio formula as in the BLOSUM62 substition matrix. For two amino acids x and y, the substitution score $s(x, y)$ is

$$s(x, y) = \log \left(\frac{p(x, y)}{q(x)q(y)} \right), \tag{B.2}$$

where $p(x, y)$ is the jointly likelihood that native amino acid x is substituted by sampled amino acid y, $q(x)$ is the marginal likelihood in the native distribution, and $q(y)$ is the marginal likelihood in the sampled distribution.

Calibration. Calibration curves examines how well the probabilistic predictions of a classifier are calibrated, plotting the true frequency of the label against its predicted probability. When computing the calibration curve, for each amino acid, we bin the predicted probabilities into 10 bins and then compare with the true probability.

Placement of hydrophobic residues. We define the amino acids IVLFCMA as hydrophobic residues, and inspect the distribution of solvent accessible surface area for both hydrophobic residues and polar (non-hydrophobic) residues. Solvent accessible surface area calculated with the Shrake-Rupley ("rolling probe") algorithm from the biotite package [75] and summed over all atoms in each amino acid. All models have similar distributions of accessible surface area for hydrophobic residues, also similar to the distribution in native sequences (Figure B.8).

Sampling speed. We profile the sampling speed with PyTorch Profiler, averaging over the sampling time for 30 sequences in each sequence length bucket on a Quadro RTX 8000 GPU with 48GB memory. For the generic Transformer decoder, we use the incremental causal decoding implementation in fairseq [95]. For GVP-GNN, we use the implementation from the gvp-pytorch GitHub repository.

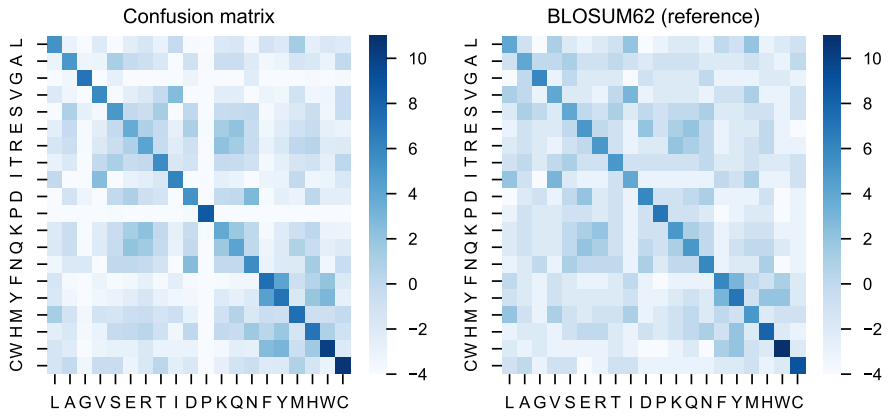

Figure B.6: Confusion matrix between native sequence and sampled sequences from the model, compared to BLOSUM62 as reference.

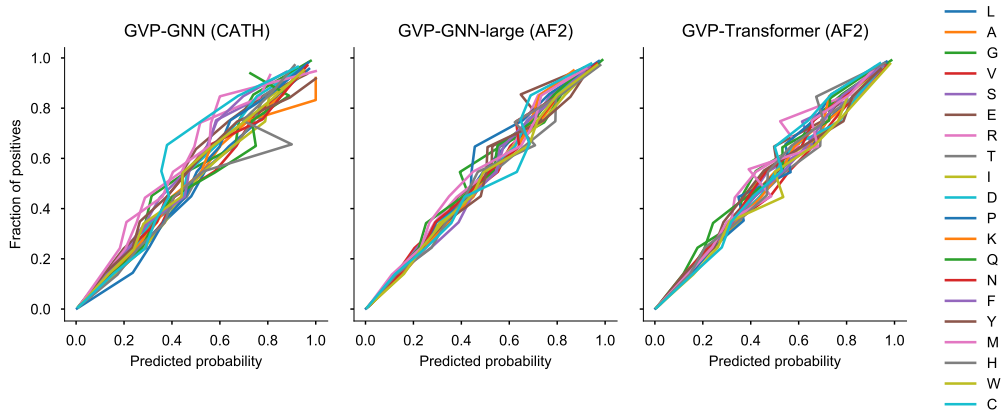

Figure B.7: Calibration.

	Average sampling time per sequence				
Sequence length	≤ 100	$100 - 200$	$200 - 300$	$300 - 400$	$400 - 500$
GVP-GNN (3 layers)	3.7s	9.3s	20.8s	76.9s	150.3s
GVP-GNN-large (8 layers)	6.7s	11.8s	47.5s	90.3s	168.8s
GVP-Transformer (8 layers)	1.5s	2.6s	9.0s	16.2s	26.0s

Table B.9: Average time required for sampling one sequence, using open source implementation of GVP-GNN and open source implementation of Transformer from fairseq [95].

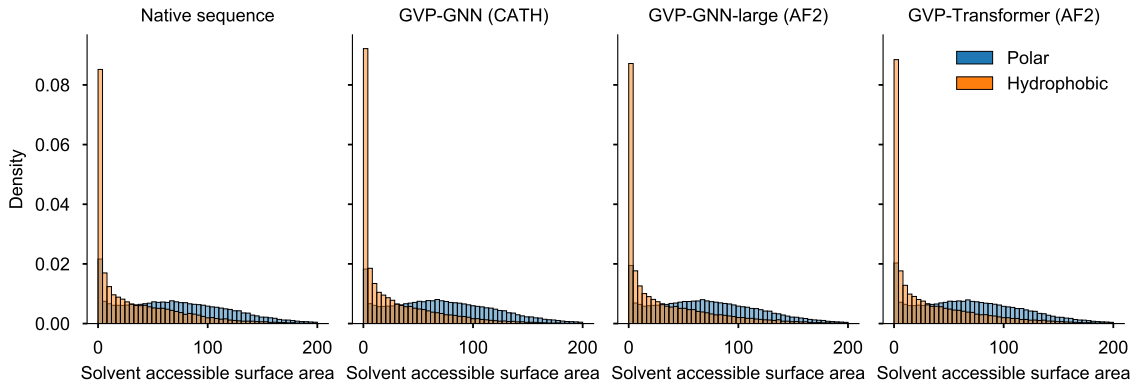

Figure B.8: The majority of hydrophobic residues are buried, following a long tail accessible surface area distribution as in native sequences.