Cell Systems

Article

Informed training set design enables efficient machine learning-assisted directed protein evolution

Bruce J. Wittmann,[1] **Yisong Yue,**[2] **and Frances H. Arnold**[1,3,4,*]
[1]Division of Biology and Biological Engineering, California Institute of Technology, MC 210-41, 1200 E. California Blvd., Pasadena, CA 91125, USA
[2]Department of Computing and Mathematical Sciences, California Institute of Technology, MC 305-16, 1200 E. California Blvd., Pasadena, CA 91125, USA
[3]Division of Chemistry and Chemical Engineering, California Institute of Technology, MC 210-41, 1200 E. California Blvd., Pasadena, CA 91125, USA
[4]Lead contact
*Correspondence: frances@cheme.caltech.edu
https://doi.org/10.1016/j.cels.2021.07.008

SUMMARY

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

INTRODUCTION

Enzyme engineering has revolutionized multiple industries by making chemical processes cheaper, greener, less wasteful, and, overall, more efficient. Enzymes and other proteins are engineered by searching the protein fitness landscape (Box 1), a surface in a high-dimensional space that relates a desired function ("fitness") to amino acid sequences (Smith, 1970; Romero and Arnold, 2009). Exploring this landscape is extremely challenging: the search space grows exponentially with the number of amino acid positions considered, functional proteins are extremely rare, and experimental screening of proteins can be resource intensive, with researchers often limited to testing a few hundred or thousand variants. Directed evolution (DE) can overcome these challenges by employing a greedy local search to optimize protein fitness (Arnold, 2018). In its lowest-screening-burden form (hereafter referred to as "traditional DE"), DE starts from a protein having some level of the desired function, then iterates through rounds of mutation and screening, where in each round single mutations are made (e.g., by site-saturation mutagenesis) to create a library of variants and the best variant is identified and fixed; iteration continues until a suitable level of improvement is achieved (Figure 1A).

By focusing on single mutations rather than combinations of mutations, traditional DE can be used to optimize protein fitness with a low screening burden. The process is highly effective when the beneficial effects of mutations made at different sequence positions are additive; however, focusing on single mutants ignores the codependence of mutations (epistasis) (Miton and Tokuriki, 2016; Starr and Thornton, 2016). Epistasis is commonly observed, for example, between residues close together in an enzyme active site or protein binding pocket, where mutations often affect function. Epistatic effects can decrease the efficiency of DE by altering the shape of the protein fitness landscape. Specifically, epistasis can alter gradients on the fitness landscape to make the route to a global optimum (Box 1) very long (Kaznatcheev, 2019), or it can introduce local optima (Box 1) at which traditional DE can become trapped (Figure 1B). Both lower the average fitness that can be achieved for a given screening burden. The only way to account for epistasis during optimization is to evaluate and fix combinations of mutations, bypassing the path-dependence of traditional DE. Due to limited screening capacity, however, this is intractable for most protein engineering projects.

Box 1. Glossary

A glossary of common technical terms used throughout this manuscript.

Active learning: a machine learning strategy where a model can propose a set of data points for a researcher to collect. The researcher then collects the data and feeds them into the model, which in turn recommends a new set of data to collect. This cycle continues until a desired engineering goal is achieved.

Cross-validation error: the mean validation error (see below) of all folds tested in k-fold cross-validation (see below).

Cross-validation indices: an "index" in programing is an integer that gives the position of an object in a list-like ordering of data. In this work, we use the term "cross-validation indices" to refer to a set of indices that give the positions of all datapoints used in each fold of k-fold cross-validation (see below). Two experiments using the same cross-validation indices thus split the data in the same way for performing k-fold cross-validation.

Encoding: machine learning models operate on numerical inputs. As a result, non-numerical inputs (e.g., a protein sequence) must first be represented—or, "encoded"—using a set of numbers. In this work, we use the term "encoding" to refer to (1) the general set of features (see below) that are used for representing protein sequences, (2) a vector of features that describes a specific protein sequence, or (3) the process of converting a non-numerical input to a numerical one (e.g., encoding produces a vector of features from a protein sequence).

Feature: a numerical value that describes some aspect of a non-numerical input to a machine learning model. A set of features makes up an encoding (see above).

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

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

K-fold cross-validation: the performance of a machine learning model is typically described using both a training error and validation error. Training error represents the ability of a model to predict the correct values for the training data and is directly used to learn model parameters. Validation error represents the ability of the model to predict the correct values for data not used to train it. Validation error is used to select the optimal model architecture for a task to avoid selecting an architecture that has obtained good training error by overfitting to noise or other idiosyncrasies in the training data. When working with limited data, however, it is often undesirable to set aside a set of data for validation error calculation, as that data will by definition not be used for training. In such a case, k-fold cross-validation can be used. In this procedure, data are split into k chunks, or "folds." Then, a model is trained using k-minus-1 folds and validation error is calculated on the held-out fold. A new model is then instantiated and trained on a different combination of k-minus-1 folds, again calculating a validation error on the held-out fold. The procedure iterates until all folds have been used for calculating validation error and a mean "cross-validation error" is returned. Model architectures that achieve good cross-validation error are assumed to be the most effective for a given task.

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

Learned embedding: for the purposes of this work, a "learned embedding" is an automatically learned encoding (see above) derived from unlabeled data (see below).

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

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

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

Model ensemble: a set of models, often with different model architectures (see above).

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

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

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

Training set: the data used to train a machine learning model.

(Continued on next page)

Box 1. *Continued*

Unlabeled data: data that consists of x-values alone. For proteins, a dataset consisting of protein sequence data alone would be considered unlabeled.
Validation error: a value that represents the ability of a model to predict the correct values for the data not used to train it.
Validation set: data that is held aside during training of a machine learning model. After training, the validation set is used to calculate a validation error (see above) and evaluate how effectively the model learned to generalize beyond the training data.
Zero-shot prediction: for the purposes of this work, “zero-shot prediction” refers to a prediction made using a model that can be trained or used without the need for additional experimental collection of data (i.e., collecting additional labeled data).

Increasingly, machine learning (ML) is being used to ease experimental screening burden by evaluating proteins *in silico* (Iuchi et al., 2021; Li et al., 2019, 2021; Mazurenko et al., 2020; Siedhoff et al., 2020; Sinai and Kelsic, 2020; Wittmann et al., 2021; Xu et al., 2020; Yang et al., 2019). Data-driven ML models learn a function that approximates the protein fitness landscape, and they require little to no physical, chemical, or biological knowledge of the problem. Once trained, these models are used to predict the fitness of previously unseen protein variants, dramatically increasing screening capacity and expanding the scope of the protein fitness landscape that can be explored by replacing expensive laboratory experimentation with *in silico* screening. We recently demonstrated a machine learning-assisted directed evolution (MLDE) strategy for navigating epistatic fitness landscapes that cover a small number of amino acid sites (Wu et al., 2019). MLDE works by training an ML model on a small sample (10^1–10^2) of variants from a multi-site simultaneous-saturation mutagenesis (“combinatorial”) library, each with an experimentally determined fitness (i.e., a model is trained using a small sample of sequence data labeled by fitness); the model is then used to predict the fitness of all remaining variants in the combinatorial library (10^4–10^5), effectively exploring the full combinatorial space. Combinations with the highest predicted fitness are experimentally evaluated, the best combination is fixed, and another round of MLDE is started at a new set of positions (Figure 1C). The iterative nature of MLDE is identical to that of traditional DE, but by evaluating and fixing multiple cooperative mutations, MLDE avoids some local fitness traps or long paths to the global optimum for each combinatorial library.

Our original MLDE work serves as a baseline, as it did not explore the many design considerations of MLDE (Figure 1C, bold and underlined questions) (Wu et al., 2019). Two notable considerations are (1) the choice of encoding (Box 1) strategy and (2) the handling of low-fitness variants in combinatorial libraries. Protein sequences must be numerically encoded to be used in ML algorithms, and the choice of encoding will affect the outcome of learning. In our original implementation, we used a one-hot (Box 1) encoding scheme, which is a simple categorical encoding that captures no information about the biochemical similarities and differences of amino acids. Mutating an amino acid to a similar one (in terms of size, charge, etc.) is less likely to affect protein fitness than mutating it to a very different one, however, and this knowledge can be transferred into ML models via the encoding strategy. The effectiveness of an ML model is also determined by the information content of the data used to train it, and so the choice of variants to use for the training stage of MLDE is important. Combinatorial libraries tend to be enriched in zero- or extremely low-fitness variants, particularly when constructed in regions critical to protein function like an enzyme active site (Arnold, 2011; Bloom et al., 2005; Romero et al., 2013). These “holes” provide minimal information about the topology of the regions of interest in a fitness landscape (i.e., they provide no information about regions with functioning proteins and no information about the *extent* to which different mutations affect fitness) and can bias ML models to be more effective at predicting low-fitness variants than high-fitness ones, the opposite of our goal. In our original implementation, we opted to sample randomly from full combinatorial spaces to generate training data with high sequence diversity. Because combinatorial landscapes tend to be dominated by holes, however, this random draw primarily returned sequences with extremely low or zero fitness, resulting in training data that, despite containing diverse sequences, was information poor.

In this work, we evaluate various design considerations by simulating MLDE on the empirically determined four-site combinatorial fitness landscape (total theoretical size of $20^4 = 160,000$ protein variants) of protein G domain B1 (GB1) (Figure 1D) (Wu et al., 2016). This landscape contains multiple fitness peaks (the routes to which are not always direct) and is heavily populated by zero- and low-fitness variants (92% have fitness below 1% of that of the global maximum), and thus not only presents an ideal testing ground in which to compare the abilities of traditional DE and MLDE to navigate epistatic fitness landscapes, but also serves to test the ability of ML methods to navigate hole-filled regions of protein fitness landscapes. We begin by evaluating a number of alternate encoding strategies to one-hot, including physicochemical encodings (encodings that capture physical and chemical properties of different amino acids) and learned embeddings (Box 1) derived from eight different natural language processing models (which are encodings extracted from ML models that represent physicochemical and contextual information about different amino acids—more background is provided in the relevant section) (Elnaggar et al., 2020; Georgiev, 2009; Rao et al., 2019, 2021; Rives et al., 2021). Next, we demonstrate how integration of models and training procedures better tailored for protein fitness landscapes into the workflow can improve MLDE performance (Bai et al., 2018; Chen and Guestrin, 2016; Zhang et al., 2020). We then show the importance of reducing uninformative holes in MLDE training sets (Box 1) and propose integrating a form of zero-shot prediction (i.e., prediction of variant fitness prior to data collection; Box 1) into the MLDE pipeline to generate more informative training data. We call the general strategy of running MLDE with training sets designed to avoid holes “focused training MLDE” (ftMLDE). We next evaluate the effectiveness of a number of zero-shot strategies for designing training data for ftMLDE applied to GB1, including state-of-the-art strategies that leverage local evolutionary information from multiple sequence alignments

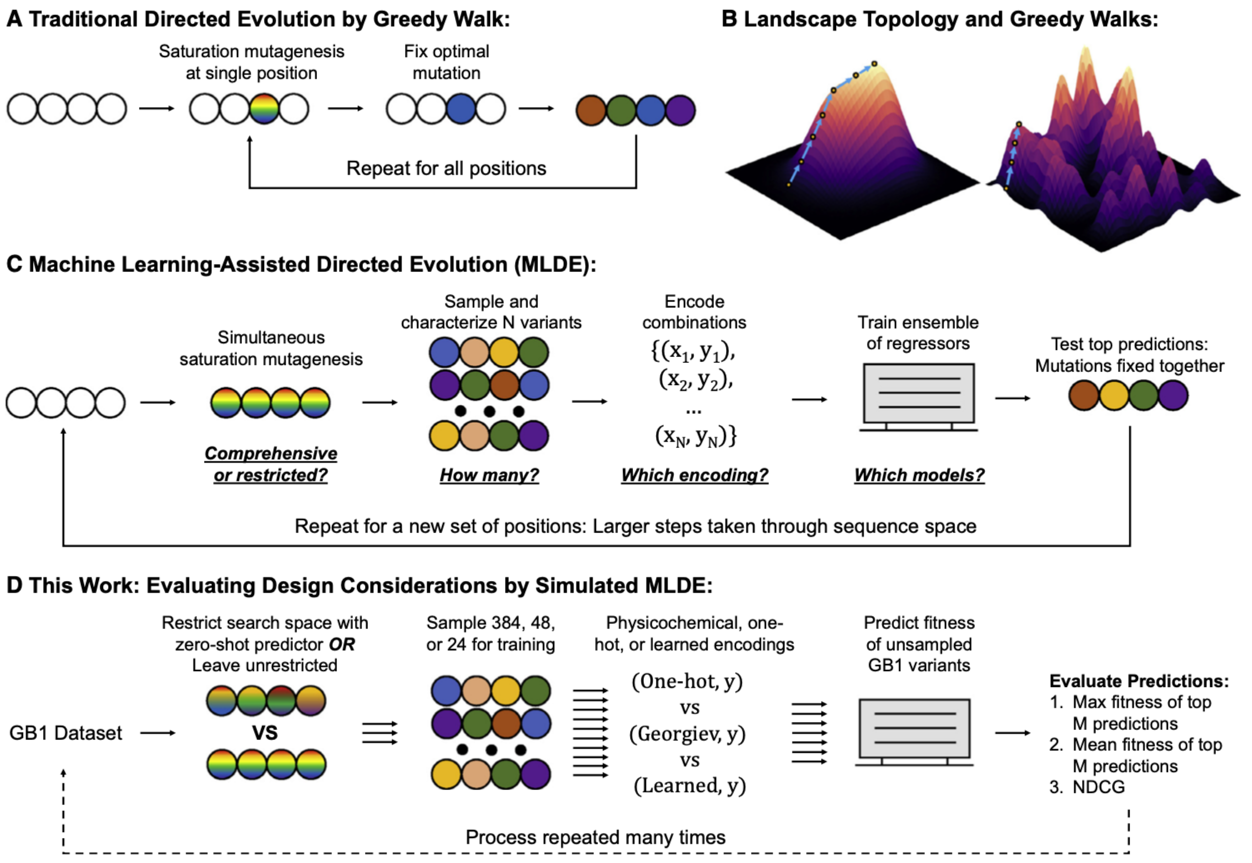
A Traditional Directed Evolution by Greedy Walk:
Saturation mutagenesis at single position
Fix optimal mutation
Repeat for all positions
B Landscape Topology and Greedy Walks:
C Machine Learning-Assisted Directed Evolution (MLDE):
Simultaneous saturation mutagenesis
Comprehensive or restricted?
Sample and characterize N variants
How many?
Encode combinations
{(x1, y1), (x2, y2), ... (xN, yN)}
Which encoding?
Train ensemble of regressors
Which models?
Test top predictions: Mutations fixed together
Repeat for a new set of positions: Larger steps taken through sequence space
D This Work: Evaluating Design Considerations by Simulated MLDE:
GB1 Dataset
Restrict search space with zero-shot predictor OR Leave unrestricted
VS
Sample 384, 48, or 24 for training
Physicochemical, one-hot, or learned encodings
(One-hot, y) vs (Georgiev, y) vs (Learned, y)
Predict fitness of unsampled GB1 variants
Evaluate Predictions:
1. Max fitness of top M predictions
2. Mean fitness of top M predictions
3. NDCG
Process repeated many times

Figure 1. Directed evolution strategies and the effects of landscape topology

(A) Directed evolution (DE) by single-mutation greedy walk (“Traditional DE”). In this approach, mutations are fixed iteratively by walking up the steepest fitness gradient.

(B) Smooth (left) versus rugged (right) fitness landscapes. A smooth fitness landscape contains a single fitness maximum, so traditional DE is guaranteed to eventually reach the global optimum, though the number of steps needed will depend on the topology of the peak. A rugged fitness landscape contains multiple fitness maxima. Traditional DE is only guaranteed to reach a local fitness optimum here; the maximum achieved will depend on the starting protein variant and the order in which positions are chosen for mutagenesis and testing.

(C) Machine learning-assisted directed evolution (MLDE). In this approach, standard molecular biology techniques are used to construct a “combinatorial library” by making mutations at multiple positions simultaneously (e.g., through use of “NNK” degenerate primers). Samples are drawn from this library (e.g., picking colonies from a plate), sequenced, expressed, assayed, and then used to train an ensemble of regressors. This ensemble is used to predict which combinations not seen in the initial draw will have the highest fitness, which are then constructed and tested experimentally. Because the best mutations are fixed simultaneously, MLDE operates in a path-independent manner, so the global optimum of a combinatorial space can be achieved regardless of the starting point. Once mutations are fixed for a given set of positions, a new set is chosen and the procedure is repeated, allowing for larger, more efficient steps through sequence space. The MLDE procedure has many design considerations, which are highlighted as questions under each step.

(D) The simulation procedure used throughout this study to evaluate improvements to the MLDE workflow, with the tests performed to evaluate the different design considerations given above each step. The simulation procedure is repeated many times using data from the GB1 landscape. The effectiveness of a simulated MLDE experiment is determined by (1) evaluating the max and mean true fitness of the top M variants according to predicted fitness and (2) calculating the normalized discounted cumulative gain (NCDG) over all predictions in the simulation.

(MSAs) (Hopf et al., 2017; Riesselman et al., 2018), a “masked-token prediction” strategy that leverages global sequence information derived from large sequence databases (Devlin et al., 2018; Elnaggar et al., 2020; Rao et al., 2021; Rives et al., 2021), and predicted ΔΔG of protein stability upon mutation. We then use the effective zero-shot predictors to generate information-rich training data. Finally, using this training data, we test the effect of training set size, the zero-shot predictor used for training set construction, and protein encoding on the outcome of ftMLDE.

In all, we found that, while using more informative encodings and models better tailored for combinatorial fitness landscapes could improve MLDE outcome, the most important design consideration was training set design, with ftMLDE generally showing improved identification of the GB1 global fitness maximum compared to MLDE. Our most effective combination of MLDE design considerations—384 training points chosen using predicted ΔΔG as the zero-shot predictor and with sequences encoded using embeddings derived from the recently published MSA Transformer (Rao et al., 2021)—successfully identified the GB1 global maximum in 99.70% of 2,000 simulated ftMLDE experiments. This represents an 81.1-fold improvement over traditional DE (which achieved the global maximum 1.23% of the time in simulated experiments) and at least a 12.2-fold improvement over our originally published method (which achieved the global optimum 8.17% of the time with a screening burden of 570 total variants—90 more than were used in our work).

This paper describes improvements to our original MLDE method. It also highlights (1) the importance of considering the unique attributes of fitness landscapes when applying ML to protein engineering problems, (2) the importance of informative training set design for building effective ML models in protein engineering, and (3) how tools developed across a variety of protein engineering domains can be combined into a cohesive, highly efficient engineering pipeline. To improve access to such a pipeline, we introduce the MLDE software package, made available on the Arnold Lab GitHub (https://github.com/fhalab/MLDE). Designed to be accessible to non-ML and non-computational experts, this repository contains Python scripts that allow execution of MLDE and ftMLDE on arbitrary combinatorial fitness landscapes, thus enabling wet-lab application.

RESULTS

MLDE procedure, simulated MLDE, and evaluation metrics

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

Throughout this work, we evaluate design considerations of MLDE through simulation on the empirically determined four-site combinatorial fitness landscape of protein G domain B1 (GB1). Originally reported by Wu et al., this landscape consists of 149,361 experimentally determined fitness measurements for 160,000 possible variants, where fitness is defined by both the ability of the protein to fold and the ability of the protein to bind antibody IgG-Fc (Wu et al., 2016). To our knowledge, this landscape is the only published one of its kind (i.e., the only almost-complete combinatorial landscape where fitness is reported as scalar values amenable to training the regression models used in MLDE). By imputing the fitness of the remaining 10,639 variants and evaluating the resultant complete landscape, Wu et al. identified 30 local optima, the routes to which were often indirect (e.g., if a local optimum was four mutations away from a starting point, it would take more than four mutations to travel by single-mutation greedy walk from the starting point to the optimum). Epistatic interactions are thus highly prevalent in the GB1 landscape. The goal of simulated MLDE is to mimic what we would observe had we performed thousands of MLDE experiments on GB1. Thus, to ensure that our simulations match what would have been observed experimentally had our simulated experiments actually been performed, we do not use the variants with imputed fitness in this study.

A simulated MLDE experiment begins with generating training data (Figure 1D). Here, a small set of variants is drawn from the GB1 landscape (values of 24, 48, or 384 are used throughout this study) and their known fitness values are attached (the variants are labeled; Box 1). This stage of the simulation is analogous to building a combinatorial library (e.g., by using "NNK" degenerate primers to make mutations at multiple positions simultaneously), picking colonies from an agar plate, then sequencing, expressing, and assaying the variants harbored by the colonies. The training data are then fed into the MLDE pipeline and the average predictions of an ensemble of the top three models are used to rank the unlabeled variants not in the training data (148,977 or more total variants, depending on the number of samples in the training data) by predicted fitness. The quality of the returned ordering is evaluated using a combination of metrics, including (1) the max fitness of the M-highest-ranked variants, (2) the mean fitness of the M-highest-ranked variants, and (3) the ranking metric "normalized discounted cumulative gain" (NDCG) (evaluation metrics) (Järvelin and Kekäläinen, 2002). Whenever reported, mean and max fitness achieved are normalized to the highest fitness in the unlabeled dataset and so can typically be interpreted as a fraction of the global maximum in the GB1 dataset.

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

NDCG is commonly used to assess the quality of information retrieval algorithms such as search engines, a task that parallels the goal of MLDE (Järvelin and Kekäläinen, 2002). To explain, the goal of search engines is to return a list of relatively rare, highly relevant documents identified among a population of many irrelevant ones; the most-relevant documents should be provided at the top of the list and the least relevant at the bottom. Because combinatorial fitness landscapes tend to be dominated by zero- and low-fitness variants (Arnold, 2011; Bloom et al., 2005; Romero et al., 2013), the goal of MLDE is likewise to identify high-fitness (high-relevance) protein variants among a sea of irrelevant ones; the highest-fitness variants should ideally be the ones ranked highest by MLDE. For both search engines and MLDE, more weight should be placed on correctly identifying and ranking the most relevant items than the least relevant as these are the ultimate items of interest. Indeed, NDCG provides just this type of implicit weighting, which is clear from the equation used to calculate it. The equation for NDCG is

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

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

More informative encodings can improve MLDE outcome

Protein sequences must be numerically encoded by a set of features (numerical descriptors that describe a protein sequence; Box 1) to be used in ML algorithms. Our previous implementation of MLDE used one-hot encoding, an uninformative categorical encoding strategy that captures no information about the biochemical relatedness of different amino acids. The descriptiveness of the features used for encoding can affect the outcome of learning, however, by passing in relevant information to an ML model about the similarities and differences between different datapoints. To investigate the effects of more informative encodings on MLDE, we tested encoding using physicochemical parameters as well as learned protein embeddings.

Physicochemical parameters are manually engineered features that describe amino acid qualities such as hydrophobicity, volume, mutability, etc. Encoding a protein sequence using these features provides an ML model with information on the physicochemical similarities and differences between amino acids. For instance, valine and alanine would have a more similar "hydrophobicity" score than valine and glutamate. In this work, we used the set of physicochemical parameters developed by Georgiev, which is a low-dimensional representation of over 500-amino-acid indices from the AAIndex database (Georgiev, 2009; Kawashima et al., 2008; Ofer and Linial, 2015).

Unlike manually crafted physicochemical parameters, learned protein embeddings are featurizations of protein sequences that have been *automatically* learned by ML models through a strategy known as "representation learning" (Iuchi et al., 2021; Li et al., 2021; Wittmann et al., 2021). All extant protein sequences have been selected by natural evolution to perform a function that is useful for their host organism. The goal of representation learning is to directly learn features that describe these proteins, thereby capturing a numerical encoding (an embedding) of the essence of what defines a functional and useful protein. Exactly how this learning is accomplished varies, though most strategies and models currently used are adapted from the field of natural language processing and rely on ever-growing protein sequence databases as a source of training data (UniProt Consortium, 2019; Iuchi et al., 2021; Li et al., 2021; Wittmann et al., 2021; Young et al., 2018). As with physicochemical parameters, learned protein embeddings capture the similarities and differences between specific amino acids; they also, however, capture contextual information about amino acid positions in a protein, with the exact embedding for a given amino acid changing based on the identities of other amino acids in the same protein sequence (Rives et al., 2021; Vig et al., 2020).

A number of studies have been performed to train models for the production of learned protein embeddings (Iuchi et al., 2021). Given a protein sequence, such models output a matrix of values that are then used to encode the protein. In this work, we test the effectiveness of learned protein embeddings generated from a variety of models of different sizes and architectures made available in the tasks assessing protein embeddings (TAPE), evolutionary scale modeling (ESM), and ProtTrans GitHub repositories (Elnaggar et al., 2020; Rao et al., 2019, 2021; Rives et al., 2021). The models tested from TAPE were trained using 30 million protein sequences from the Pfam database (El-Gebali et al., 2019) and have varied architectures, including a transformer architecture ("TAPE Transformer") (Devlin et al., 2018; Vaswani et al., 2017), three separate LSTM-based architectures ("LSTM," "UniRep," and "Bepler") (Alley et al., 2019; Bepler and Berger, 2019; Hochreiter and Schmidhuber, 1997), and a dilated residual network architecture ("ResNet") (Yu et al., 2017)); all models in TAPE are defined by around 38 million learnable parameters.

Larger models than those in TAPE trained on more sequences can potentially learn a richer representation of protein sequences (Brown et al., 2020; Rives et al., 2021). To test this potential effect of model and training set size on the quality of learned embeddings for MLDE, we also investigated embeddings generated from the state-of-the-art models "esm1b_t33_650M_UR50S" (hereafter referred to as "ESM1b") from the ESM repository as well as "ProtBert-BFD" from the ProtTrans repository. Both of these models have a transformer architecture. ESM1b is a 650-million-parameter model trained on 27.1 million sequences from the UniRef50 database (UniProt Consortium, 2019; Rives et al., 2021). ProtBert-BFD is a 420-million-parameter model trained on 2.1 billion protein sequences from the Big Fat Database (BFD) (Elnaggar et al., 2020; Steinegger and Söding, 2018). A final model investigated for learned embedding generation was the MSA Transformer, also made available in the ESM repository (Rao et al., 2021). Unlike the other models, which were all trained on protein sequences, the 100-million-parameter MSA Transformer was trained on 26 million protein MSAs; the learned embeddings from the MSA Transformer are thus generated from an MSA of the target protein rather than the target protein sequence alone. MSAs more directly represent information relevant to protein engineering: specifically, related sequences aligned to a reference provide evidence for what mutations are and are not allowed at given positions. We included the MSA Transformer to test if the additional information provided by embeddings generated from an MSA could lead to an improved MLDE outcome.

For each encoding considered, we performed 2,000 rounds of MLDE simulations at three different training set sizes. The training data, cross-validation indices (i.e., the different folds used for measuring a cross-validation error; Box 1), and random seeds (values that enable reproducible random number generation; Box 1) were kept the same for each encoding strategy in each simulation. For a given simulation, the training data consisted of either 384, 48, or 24 GB1 variants drawn at random from the *comprehensive* (consisting of all 149,361 possible GB1 variants) landscape—only the choice of encoding and training set size were considered as design considerations in these experiments. If 384 variants were used for training, the top 96 predictions were tested; if 48 variants were used for training, the top 32 predictions were tested; if 24 variants were used for training, the top 56 predictions were tested (encoding comparison simulations). Training using 384 samples and testing 96 predictions evaluates simulations on a scale that approximates the typical experimental screening burdens for standard DE approaches (Wu et al., 2019). Because the ultimate goal of using ML in protein engineering is to reduce or eliminate the number of protein variants that must be experimentally characterized, the ability to train an ML model using limited data is also valuable (Biswas et al., 2021). Training using 24 or 48 samples evaluates the effectiveness of each encoding in this "low-N" setting. We tested 56 or 32 predictions, respectively, to match the total screening burden (80 variants) of an idealized traditional DE pipeline over a four-site landscape where all 20 amino acids at each position are deterministically evaluated. We note that, due to the cost of synthesizing variants individually, deterministic evaluation of mutations is rarely performed, and researchers instead opt to stochastically sample from pools of mutants (thus raising the required screening burden above 80). The total screening burden of deterministic traditional DE does provide, however, a reasonable "low-N" threshold for MLDE.

Violin plots showing the results of the simulated MLDE experiments are provided in Figure 2; summary statistics are provided in Data S1, and a pairwise comparison of encoding effectiveness over all simulations is provided in Data S2; we also provide additional figures on the GitHub repository associated with this work that plot the pairwise encoding comparisons. In all, these results show that using a more informative encoding than one-hot can result in an improved MLDE outcome, but not always and depending on the metric and screening burden used to measure MLDE effectiveness. The only two encoding strategies to consistently show at least marginal improvement over the one-hot baseline regardless of metric and screening burden were physicochemical (Georgiev) parameters and learned embeddings from the MSA Transformer. At a training size of 384, NDCG was the only evaluation metric to consistently suggest that more informative encodings improve MLDE outcome. For the max fitness achieved, simulations run using Georgiev parameters and learned embeddings from the MSA Transformer tended to achieve marginally higher max fitness than those run using one-hot encodings; simulations run using all other learned embeddings tended to achieve the same if not a slightly lower one. For the mean fitness achieved, simulations run using the embeddings from the Bepler model as an encoding strategy slightly underperformed one-hot, those using embeddings from UniRep and ProtBert-BFD performed comparably, and those using all other encoding strategies tended to achieve a higher mean fitness than one-hot.

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

Our results largely agree with recent work suggesting that training ML models using existing learned protein embeddings yields marginal improvement at best compared with using simpler encodings such as one-hot or physicochemical parameters (Hsu et al., 2021; Shanehsazzadeh et al., 2020). Unlike previous works, however, which found that learned embeddings tended to be superior to simpler strategies in the low-N regime (Biswas et al., 2021; Shanehsazzadeh et al., 2020), we found that simpler strategies remained competitive regardless of training set size. It is possible that taking an "evotuning" strategy like that of Biswas et al., where embedding models are further trained on sequences more closely related to the target protein (i.e., GB1), could improve the performance of learned embeddings in the low-N regime (Biswas et al., 2021); however, as will be discussed in greater detail in later sections, this possibility is currently untestable due to the limited availability of GB1 homologs in existing sequence databases.

We also found minimal benefit in using embeddings derived from the larger models trained on larger corpora of protein sequences. For instance, when trained with 384 samples, simulations run using embeddings from ESM1b only slightly outperformed those run using embeddings from the TAPE transformer, despite the ESM1b model being ~17-fold larger; in the low-N regime, simulations run using embeddings from ESM1b underperformed those run using encodings from the TAPE transformer. Likewise, regardless of training set size, simulations run using embeddings from ProtBert-BFD often underperformed many of the TAPE models, despite ProtBert-BFD being ~11-fold larger and trained using ~70-fold more protein sequences.

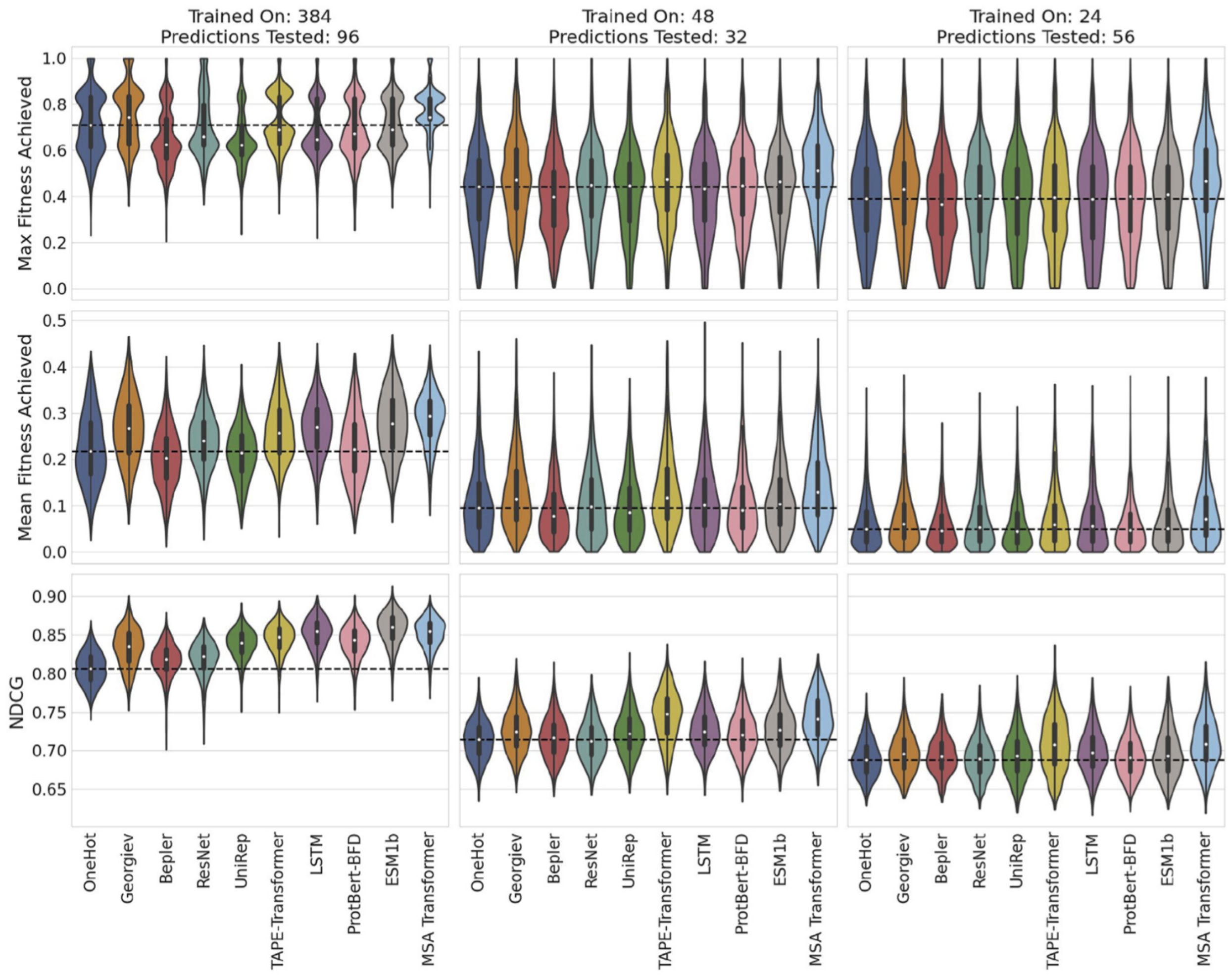

Figure 2. More informative encodings can improve MLDE outcome: Results of simulated MLDE comparing ten different encoding strategies at three different screening burdens

Note that, for the sake of computational efficiency, 19 of the 22 inbuilt MLDE models were in the ensemble trained for simulations using the large TAPE transformer-, the MSA Transformer-, ESM1b-, ProtBert-BFD-, UniRep-, and LSTM-derived encodings, while 22 were in the ensemble for all others. Each column of plots gives the results of a different screening burden. Each row of plots gives the results for a different summary metric. Rows and columns share the same axes. Each violin represents the results of 2,000 simulated MLDE experiments, and the dashed line represents the median summary value of simulations run using one-hot encoding. In general, whether or not an encoding strategy outperformed the one-hot baseline depended on the screening burden tested and summary metric evaluated. The only two encoding strategies to consistently show at least marginal improvement over the one-hot baseline regardless of metric and screening burden were physicochemical (Georgiev) parameters and learned embeddings from the MSA Transformer. For summary statistics of simulation results, see also Data S1. For pairwise comparisons of simulation results, see also Tables S1–S4, Data S2, and additional figures at the GitHub associated with this work.

Indeed, the most effective model for generating learned embeddings was the MSA Transformer, which is 1/6 the size of ESM1b and 1/4 the size of ProtBert-BFD. As mentioned above, the MSA Transformer was trained on—and generates embeddings using—MSAs rather than protein sequences. It is possible that the additional information provided by the MSA yields more effective learned embeddings for MLDE, though this is impossible to conclude working off of just the GB1 dataset. It does stand to reason, however, that models and data sources that more directly represent information known to be important for protein function could lead to embeddings that are more informative for MLDE. Developing such data sources and models is a potentially valuable avenue for future research.

Models/training procedures more tailored for combinatorial fitness landscapes can improve MLDE predictive performance

Many of the learned embeddings used in the previous section are extremely high dimensional, with the largest (LSTM) describing each combination of four amino acids with 8,192 features (Table S1). To better handle the high dimensionality introduced by learned embeddings, in this new implementation of MLDE we added two 1D convolutional neural network (CNN) architectures to the ensemble of models trained, one with a single convolutional layer and another with two (inbuilt models). CNNs apply sliding windows (“convolutions”) over structured, high-dimensional data, relying on spatial dependencies between elements of the input data to extract the most-relevant high-level features (Jiang and Zavala, 2021; Rawat and Wang, 2017). For instance, CNNs are often applied to image processing tasks, where sliding 2D windows are used to extract high-level features by aggregating information from local groupings of pixels. CNNs can also be applied, however, to sequential data such as protein and DNA sequences. When applied to proteins, the sliding windows are 1D (hence, "1D CNN") rather than 2D, and are applied over the protein sequence to extract high-level features by aggregating information from nearby members of the sequence (Bai et al., 2018; Jaganathan et al., 2019; Xu et al., 2020). The practice of using a sliding window to extract or aggregate information from sequences or sequence alignments has been used in bioinformatic analyses for decades (Proutski and Holmes, 1998; Tajima, 1991; Zhu et al., 2020). Whereas sliding windows have historically been used to extract specific, human-defined information, the sliding windows of 1D CNNs automatically learn the aggregate information most relevant for relating a sequence to a label (e.g., the high-level features that relate protein sequence to fitness). Recent evidence suggests that 1D CNNs are a particularly effective model class for protein engineering (Xu et al., 2020). We found that 1D CNNs could be beneficial for MLDE, but that the specific architecture of the 1D CNN and training points used to train it were important. For instance, when trained with 384 training points, the two-layer 1D CNN was consistently among the top-ranking models in terms of cross-validation error during training, particularly for higher-dimensional encodings (Table S1). The same could not be said, however, for the single-layer 1D CNN or the two-layer 1D CNN trained with less data (Tables S1 and S2).

In addition to 1D CNN architectures, we also integrated XGBoost models trained with the Tweedie regression objective to better handle the zero-inflated nature of fitness landscapes (Chen and Guestrin, 2016; Yang et al., 2018; Zhou et al., 2020). XGBoost is a Python package that implements the gradient boosting technique, which, at a high level, is a strategy of combining multiple weak predictors (multiple weak models) to create a more effective predictor (Chen and Guestrin, 2016). Gradient-boosted Tweedie regression was developed to handle regression for datasets with zero-inflated labels (Yang et al., 2018; Zhou et al., 2020). Because most mutations are deleterious to activity or stability, as more mutations are made to a protein, the probability that it will still fold and function drops (Bloom et al., 2005). The result is that combinatorial fitness landscapes tend to be dominated by proteins with zero or extremely low fitness (Arnold, 2011; Romero et al., 2013), something that is highlighted by the distribution of fitness for GB1 (Figure 3A). Training data drawn from combinatorial fitness landscapes will thus also have an overabundance of zeros, which can bias ML models to be more effective at predicting low-fitness variants than high-fitness ones. To test if implementing the Tweedie regression objective could improve the effectiveness of XGBoost models in MLDE, we included XGBoost models trained with both the Tweedie and default (root mean squared) training objectives in the ensemble of models trained in the simulations discussed in the previous section. We found that models trained with the Tweedie objective on average achieved a higher NDCG than models trained with the default objective regardless of base model (the architecture of the weak predictors used by XGBoost) and encoding; however, only models with a tree base on average showed improved max and mean fitness achieved (Tables S3 and S4). Additional supplemental images plotting a pairwise comparison of the results of XGBoost simulations run with each learning objective (Box 1) can also be found at the GitHub repository associated with this work.

The challenge of holes in combinatorial fitness landscapes and the importance of informative training data

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

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

To demonstrate the concept of ftMLDE and test its effectiveness, we designed training data enriched in functional, but not the fittest, protein variants, and then used it to perform simulated ftMLDE. Because we have access to the full GB1 dataset, we can choose what data to use for training. As such, we built training sets consisting of 384 samples where 50% of the variants had fitness greater than or equal to a given threshold of either 0.011, 0.034, 0.057, or 0.080 and 50% had fitness below (high-fitness simulations). A higher fitness threshold thus meant greater fitness enrichment in the training data (greater "focus" of the training data on higher-fitness regions of the protein fitness landscape) and vice versa (Figure S1). To avoid "cheating" by inclusion of the highest-fitness variants in the training data, we also enforced a requirement that no variant in the training data had

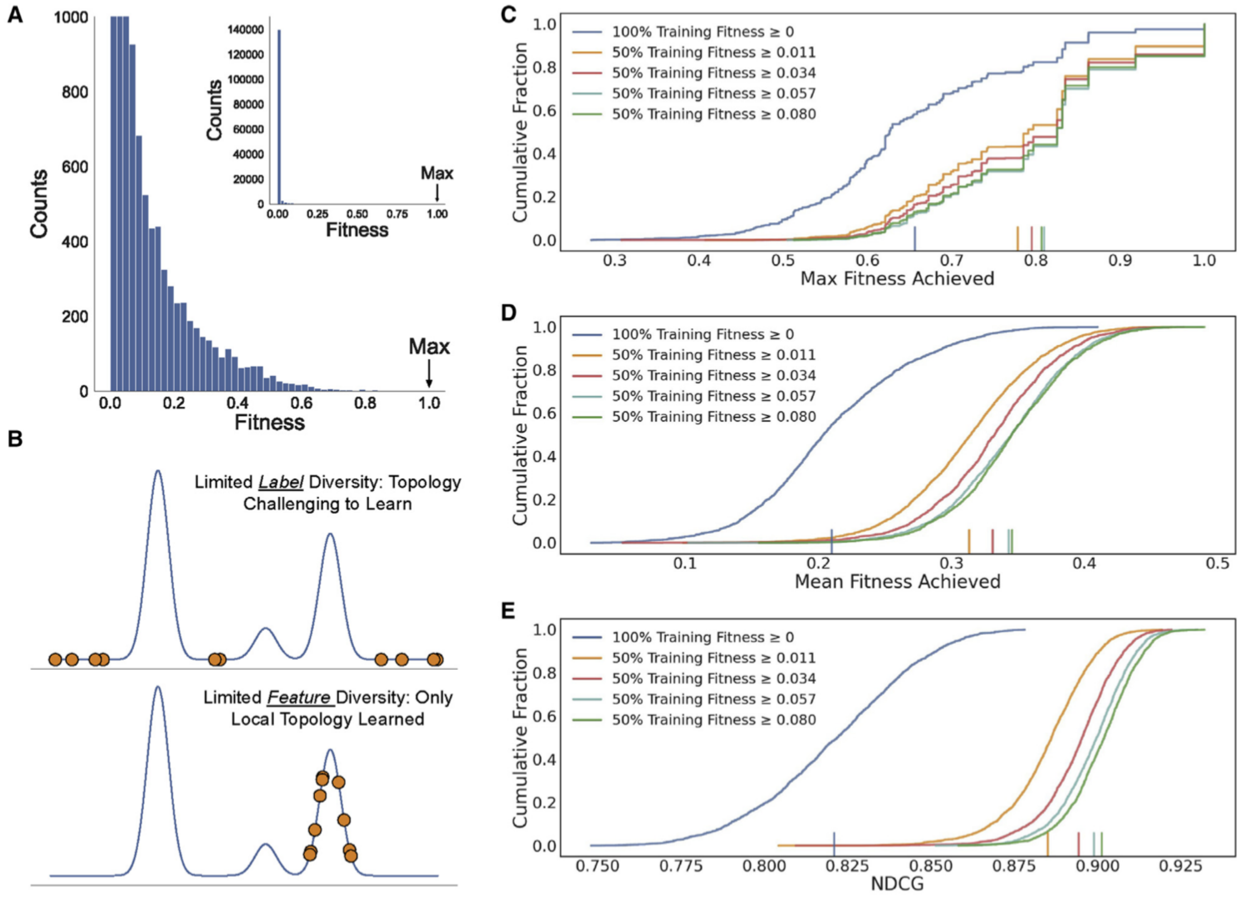

Figure 3. The challenge of holes in combinatorial fitness landscapes and the importance of informative training data

(A) The distribution of fitness in the GB1 landscape shown as a histogram. Most variants in this epistatic landscape have extremely low fitness, and the highest-fitness variants are very rare.

(B) A demonstration of the importance of diversity in both the labels and features of training data for machine learning. Learning detailed topology is challenging if the labels are not representative of it, even if sampled from diverse regions of feature space. Only local topology can reliably be learned if points are sampled from a restricted region of feature space.

(C) The maximum fitness achieved for simulated ftMLDE using training data designed to be enriched in fit protein variants. Specifically, training sets were designed such that 50% of the variants had fitness greater than or equal to a given threshold of either 0.011, 0.034, 0.057, or 0.080 and 50% had fitness below. A higher fitness threshold enforces a higher mean training fitness. All data are shown as empirical cumulative distribution functions (ECDFs); vertical lines on the x axis give the expectation value of the distribution. Each ECDF represents the results of 2,000 simulated ftMLDE experiments.

(D) The mean fitness achieved for simulated ftMLDE using training data enriched in fit variants.

(E) The NDCG for simulated ftMLDE using training data enriched in fit variants. See also Figure S1 and Table S5.

fitness greater than 34% of the global maximum. By including this upper limit, the highest-fitness variants in the GB1 landscape could only be identified from model predictions.

The results of 2,000 simulated ftMLDE experiments using training sets from each of the four considered thresholds are given in Figures 3C–3E and Table S5; also included are the results of 2,000 simulated standard MLDE experiments where training data were randomly drawn from all variants with fitness below 34% of that of the global maximum. Compared with standard MLDE, the ftMLDE simulations show improved NDCG, mean fitness achieved in the top 96 predictions, and max fitness achieved in the top 96 predictions. Training data enrichment using even the lowest fitness threshold (0.011) led to improvement in evaluation metrics, with NDCG increasing ~8%, the max fitness achieved improving ~19%, and the mean fitness achieved improving ~49%. The lowest threshold sits at just above 1% of the fitness of the global maximum, and so the improvement observed here suggests that even the weakest degree of enrichment can lead to an improvement in engineering outcome. Indeed, while further increasing the fitness threshold did further improve outcome, the degree of improvement was not as large. For instance, increasing the threshold from 0.011 to 0.080 led to a further ~2% increase in NDCG, ~4% increase in max fitness achieved, and ~10% increase in mean fitness achieved, roughly 5-fold less overall improvement compared with moving from no threshold to a threshold of 0.011. This result suggests that, although achieving a higher mean fitness in the training data is beneficial to ftMLDE, the more important factor is the elimination of holes.

Zero-shot prediction as a practical training set design strategy for ftMLDE

Of course, in practice, the full dataset for a combinatorial library would not be available as it is for GB1, otherwise there would be no point in applying MLDE in the first place. Instead, the protein variants used to build ftMLDE training data must be chosen prior to knowing their fitnesses—practical application of ftMLDE requires at least a weak predictor of protein fitness for training set design. One way to accomplish this would be to take an active learning (Box 1) approach. That is, using data from a prior round of standard MLDE performed for the same combinatorial library, a model could be trained to predict a diverse set of higher-fitness variants; these variants could then be experimentally evaluated and used to train models in a round of ftMLDE. Indeed, a strategy like this was taken by Romero et al. when evolving for improved P450 thermostability, where a classifier trained on data from one round of evolution was used to build a training dataset enriched in functional protein variants for the next round of evolution (Romero et al., 2013). While this active learning approach has proven successful, it adds an additional round of data collection to the workflow, which is undesirable. We thus chose to investigate zero-shot prediction strategies for training set design.

We define zero-shot prediction strategies as those capable of predicting protein fitness without the need for further labeled training data collection, and thus they do not affect the overall screening burden of ftMLDE. A number of zero-shot strategies exist for protein functional prediction, ranging from scoring protein variants based on evolutionary sequence conservation (Hopf et al., 2017; Ng and Henikoff, 2003; Riesselman et al., 2018) to generative modeling (Madani et al., 2020; Riesselman et al., 2019, 2018) and physics-based computational modeling (e.g., prediction of $\Delta\Delta G$ upon mutation) (Firnberg et al., 2014; Jacquier et al., 2013; Sarkisyan et al., 2016; Sirin et al., 2016; Yang et al., 2020), to name a few. The remainder of this paper is devoted to evaluating the effectiveness of different zero-shot strategies for designing training data for ftMLDE. Over the next two sections, we evaluate zero-shot strategies from each of the aforementioned overarching zero-shot classes for their ability to predict GB1 fitness. In the final section, we use the successful zero-shot predictors to demonstrate a practical application of ftMLDE to the GB1 landscape. We find that ftMLDE is superior to both standard MLDE and traditional DE, with our best ftMLDE condition achieving the global maximum in 99.70% of simulated experiments compared with 8.85% for the best standard MLDE condition and 1.23% for simulated traditional DE.

Leveraging sequence data for the design of fitness-enriched training data

Over billions of years, natural evolution has tested countless protein sequences, discarding those that were detrimental to a host organism and propagating those that were beneficial. The list of extant protein sequences represents those that survived the filters of evolution and so implicitly contains information about the evolutionary and biophysical rules that enable production of a useful protein. Driven by a combination of increased computational power and greater availability of sequence data, recent years have seen renewed effort to extract this implicit fitness information contained in sequences and use it to reduce or eliminate the amount of experimentally acquired sequence-function data needed for reliable prediction of protein fitness. All of these strategies assume that a given list of functional protein sequences is representative of a distribution of allowed protein sequences and that by learning this distribution the fitness of a new protein sequence can be inferred. Specifically, a new sequence highly likely to belong to the learned distribution is predicted to have high fitness and vice versa (Hopf et al., 2017; Riesselman et al., 2018; Wittmann et al., 2021). The simplest example of a sequence-based zero-shot strategy, for instance, is use of BLOSUM matrices, which score the likelihood of a given amino acid substitution based on observed substitution frequencies in conserved protein families (Henikoff and Henikoff, 1992). Far richer strategies than BLOSUM matrices have been developed, however, and, in this section, we test the ability of a number of them for zero-shot prediction of GB1 fitness.

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

We first tested the local sequence-based zero-shot predictors EVmutation and DeepSequence (Hopf et al., 2017; Riesselman et al., 2018). Among other requirements, the authors of these tools recommend training using an MSA with $\geq$10L (where "L" is the length of the target protein) redundancy-reduced sequences (essentially, a measure of the effective number of sequences given the diversity of those in the MSA—less diverse MSAs have a lower number of redundancy-reduced sequences) that cover the positions at which the effects of mutations are to be predicted; at least 560 redundancy-reduced sequences are thus the target for 56-amino-acid-long GB1. There are, unfortunately, few recorded sequences that are homologous to GB1, and we could at best produce an MSA with 56 redundancy-reduced sequences that covered all four positions of interest in the GB1 combinatorial landscape (alignment generation and EVmutation model training). Despite this relatively uninformative MSA, however, EVmutation still performed reasonably well as a zero-shot predictor, achieving a Spearman rank correlation coefficient (Spearman ρ) of 0.21 (Figure 4; Table S6). DeepSequence, in contrast, was less effective, achieving Spearman $\rho = 0.05$ (Table S6, EVmutation/DeepSequence calculations). These results align with an observation in the original DeepSequence publication, where DeepSequence was shown to be more susceptible to failure than EVmutation when trained on low-quality MSAs (Riesselman et al., 2018). This is not to

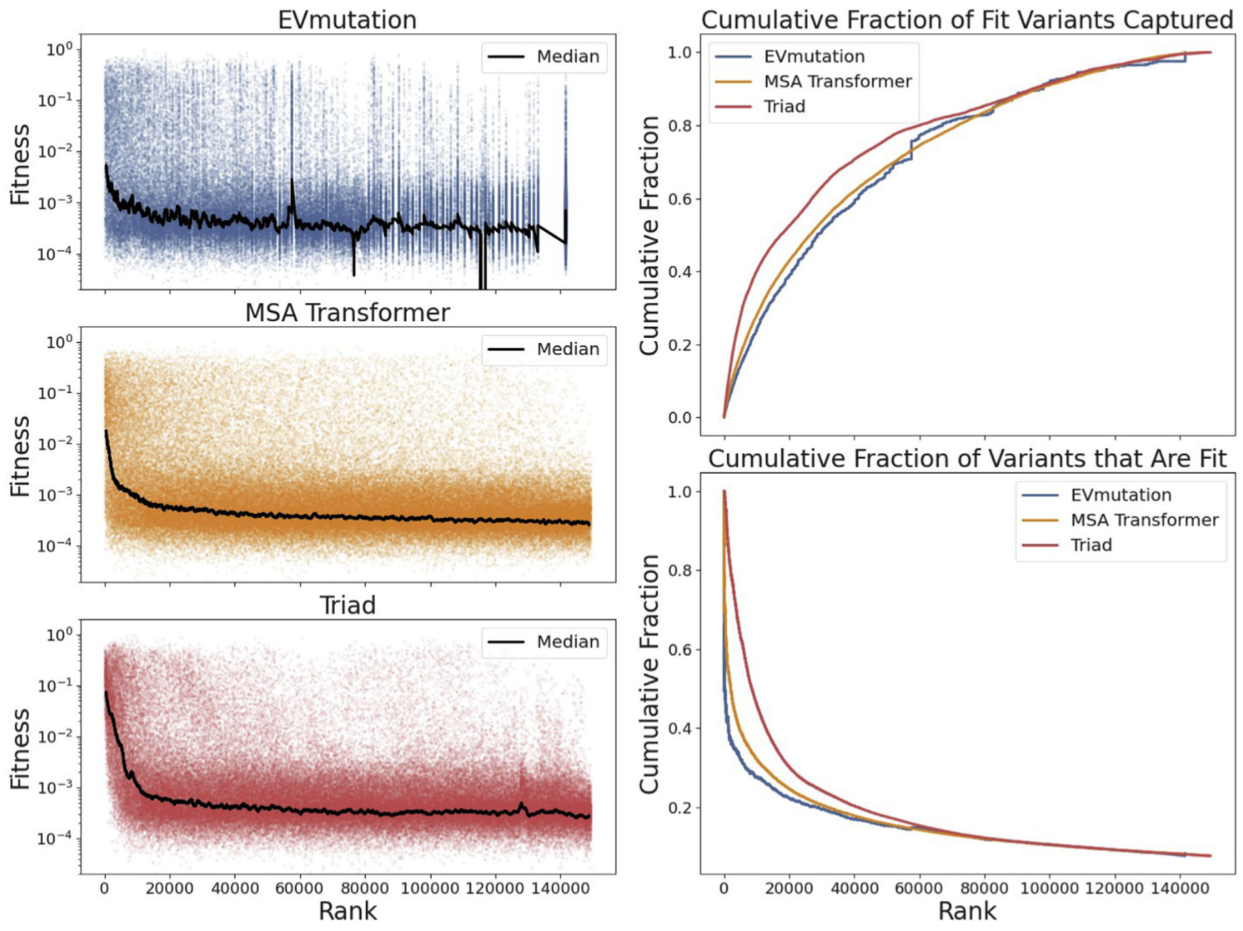

Figure 4. Zero-shot prediction for the design of fitness-enriched training data

All figures plot the predicted rank of GB1 variants (where the variants predicted to be most fit have lower rank and vice versa) against either fitness (A–C) or an alternate summary metric (D–E). In (A–C), dots are all individual variants while the black line is the sliding median (window size = 1,000) of fitness.

(A) Results of zero-shot prediction using EVmutation.

(B) Results of zero-shot prediction using a mask-filling protocol with the MSA Transformer.

(C) Results of zero-shot prediction using predicted ΔΔG from Triad with a fixed protein backbone.

(D) The fraction of all fit variants in the GB1 landscape captured up to and including a given rank. A "fit" variant is defined as one with fitness greater than 0.011 (which was the lowest threshold tested for the simulations performed with training data designed to be higher in fitness—see Figure 3).

(E) The cumulative fraction of fit variants captured up to and including a given rank. See also Tables S6 and S7 and Figures S2–S5.

say that the predictions of EVmutation were unaffected by the low-diversity GB1 MSA. The low information content of the MSA made it impossible for EVmutation to assign unique probabilities of fitness to all GB1 combinations, resulting in the coarse ranking pattern shown in Figure 4.

For global sequence-based zero-shot predictors, we tested a mask-filling protocol for each of the models made available in the ESM GitHub repository as well as the ProtBert and ProtBert-BFD models from the ProtTrans GitHub repository (Elnaggar et al., 2020; Rao et al., 2021; Rives et al., 2021). All of these models were trained using a protocol known as "masked-token prediction" (Devlin et al., 2018). When training using this protocol, a model is fed a sequence with the identities of amino acids at a fraction of its positions obscured ("masked"). Given the context of the unobscured ("unmasked") amino acids, the objective of the model is to then predict the correct original identities of the masked amino acids by modeling the probability $P(s_{masked}|s_{unmasked})$, where s is a sequence of amino acids. By repeating this procedure over millions (or billions, in the case of ProtBert-BFD) of sequences, the model learns a global sense of the distribution of allowed proteins: in particular, it learns the probability that a given combination of amino acids will occur in the context of a given sequence background. Using a trained model, masked-token prediction can be co-opted for zero-shot prediction using a "mask-filling protocol." Specifically, given a sequence with positions of interest masked, the model can be used to predict $P(s_{masked}|s_{unmasked})$ for all possible combinations of mutations at the masked positions. Combinations of mutations with higher probability are then assumed to have higher fitness as they more accurately represent the learned distribution of allowed amino acid combinations.

The models in the ESM repository, in combination with ProtBert and ProtBert-BFD, are a variety of different sizes and were trained on varying numbers of sequences, allowing us to test the effect of model capacity on the mask-filling protocol for GB1. Additionally, the ESM repository contains the MSA Transformer which, uniquely, was trained using MSAs produced for each sequence in the UniRef50 database, making it somewhat of a hybrid between a local and global sequence model (UniProt Consortium, 2019; Rao et al., 2021). We included the MSA Transformer in our mask-filling zero-shot predictions to see if the global information captured during training could make up for the limited information provided by the small GB1 MSA used for EVmutation and DeepSequence predictions.

For zero-shot prediction with a mask-filling protocol, we calculated $P(s_{masked}|s_{unmasked})$ for every combination in the GB1 landscape using either naive or conditional probability (mask-filling protocol). Note that, for all non-MSA Transformer methods, the parent GB1 sequence was used to define $s_{unmasked}$, while for the MSA Transformer, the MSA used for EVmutation (with slight additional processing, see mask-filling protocol in the STAR Methods for details) was used to define $s_{unmasked}$. The MSA Transformer thus had access to additional local evolutionary information when making mask-filling predictions. The results using both naive and conditional probability protocols for all tested models are provided in Table S7; the results using naive probability with the MSA Transformer are also depicted in Figure 4. In all cases, we found predictions made using naive probability to be slightly superior to predictions made using conditional probability. The naive probability prediction procedure more closely mimics the masked-token prediction procedure used to train the ESM and ProtBert models, providing a potential explanation for its slight superiority over the conditional prediction procedure, though a reason for this observation is not immediately clear.

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

The exception to the general failure of mask filling as a zero-shot predictor was those predictions generated by the MSA Transformer (Figure 4), which achieved a Spearman ρ of 0.24 with naive probability (0.20 with conditional). Even though this Spearman ρ is comparable to that achieved using EVmutation, it is notable that the many ties observed in the EVmutation predictions are not present in the mask-filling zero-shot predictions from the MSA Transformer, presumably due to the global information captured by the MSA Transformer during training. A mask-filling protocol using the MSA Transformer could thus be an attractive zero-shot alternative to EVmutation for proteins for which deep, high-quality MSAs cannot be produced. It must, of course, also be asked whether the underrepresentation of GB1 homologs in sequence databases leads to the failure of mask-filling zero-shot prediction by non-MSA Transformer models. Answering this question is impossible using just the GB1 landscape alone, however, and would require access to other combinatorial landscapes built in proteins with varying degrees of representation in the sequence databases used to train the ESM and ProtBert models.

Predicted ΔΔG of stabilization for the design of fitness-enriched training data

Just because a sequence motif is not represented in a sequence database does not necessarily mean that it would be detrimental to a protein's function. It is possible, for example, that a natural function that would benefit from such a motif does not exist, that evolution has not yet explored such a region of sequence space, or simply that humans have not yet sequenced a representative protein. The underlying assumption of sequence-based zero-shot strategies that evolutionarily optimized fitness correlates to a target fitness may thus not always hold. In such cases, using a zero-shot strategy such as predicted ΔΔG of stabilization upon mutation may be beneficial (Firnberg et al., 2014; Jacquier et al., 2013; Sarkisyan et al., 2016; Sirin et al., 2016; Yang et al., 2020). This approach attempts to calculate the effect of a mutation on protein stability from first principles. Based in physics, it is thus not subject to the assumption that the target fitness correlates with the fitness of existing proteins, but instead that protein stability plays a role in fitness.

The fitness of GB1 is considered to be, at least in part, a function of stability, suggesting that approaches like predicted ΔΔG of protein stability upon mutation might be an effective zero-shot predictor (Olson et al., 2014; Wu et al., 2016). Indeed, we find a correlation between single-mutant fitness data and literature GB1 ΔΔG data (|Spearman ρ| = 0.58, Figure S2A) (Nisthal et al., 2019). Wu et al. also previously presented evidence suggesting that predicted ΔΔG could be correlated to GB1 fitness (Wu et al., 2016).

To test the effectiveness of ΔΔG predictions as a zero-shot predictor for GB1 fitness, we used the Triad protein design software suite (Protabit, Pasadena, CA, USA: https://triad.protabit.com/) with a Rosetta energy function to predict the stability of each of the 149,361 GB1 variants with measured fitness, then calculated a predicted ΔΔG of stabilization for each variant relative to the parent amino acid sequence (ΔΔG calculations). Both fixed backbone and flexible backbone calculations were performed using a previously determined GB1 crystal structure (PDB: 2GI9) as a scaffold (Franks et al., 2006). The predicted ΔΔGs from each calculation correlated with literature values of experimentally determined ΔΔG values for the single mutants, though the fixed backbone calculations were more effective (Spearman ρ = 0.61 for fixed backbone, Spearman ρ = 0.42 for flexible backbone, Figures S2B and S2C). Despite both approaches having predictive power for single-mutant ΔΔG, only the fixed backbone calculations were effective at identifying GB1 variants enriched in fitness when ranking by predicted ΔΔG (Spearman ρ = 0.27, Figure 4, Table S6, Figure S3). If instead, however, the GB1 variants were ranked by root-mean-squared deviation (RMSD) of variant structures produced during flexible backbone calculations, those variants with the lowest RMSD tended to be enriched in fitness, though not as strongly as in the fixed backbone calculations (Spearman ρ = 0.06, Table S6; Figure S4).

Structurally conservative mutations are generally less likely to disrupt protein function, and so the observation that RMSD can be used for zero-shot prediction is not entirely surprising. Because fixed backbone calculations will tend to heavily penalize mutations that would require large backbone movements to stabilize, an interesting question arises over the extent to which structural conservation or accurate prediction of ΔΔG allows effective fixed backbone zero-shot prediction of fitness in GB1. If structural conservation dominates, it is possible that Triad could be used for zero-shot prediction with other combinatorial libraries in proteins where variant fitness is not related to stability, particularly when the mutated residues in question are tightly packed together and/or buried in the protein core as they are for the GB1 landscape used in this study (Figure S5). Answering this question and evaluating the generalizability of fixed backbone Triad calculations is beyond the scope of this work, but as more fully combinatorial datasets become available this question should be investigated further.

Zero-shot predictions for training set design enable highly effective ftMLDE on the GB1 landscape

As a final demonstration, we evaluated the performance of ftMLDE using GB1 training data predicted to be higher in fitness by the three successful zero-shot prediction strategies: EVmutation, mask filling using the MSA Transformer, and Triad ΔΔG calculations. To begin, we generated training data by randomly sampling 2,000 training sets of 24, 48, and 384 variants from the top 1,600 (1.1%), 3,200 (2.1%), 6,400 (4.3%), 9,600 (6.4%), 12,800 (8.6%), 16,000 (10.7%), and 32,000 (21.4%) variants as ranked by each zero-shot predictor; completely random training data (i.e., from the full landscape) were also drawn at each sample size so that standard MLDE could be performed as a control. These splits resulted in 66 total "training data types" to test (three random MLDE training data types plus 21 zero-shot ftMLDE training data types for each of three zero-shot predictors), each made up of 2,000 training sets. Predictive algorithms (zero-shot predictors included) will tend to predict that similar sequences have similar fitness, so sampling from different percentiles of the top predictions explores the exploration-exploitation tradeoff of using zero-shot predictions for training set design. In other words, sampling from a larger top percentile of the ranked variants allows greater sequence diversity in the training data (thus potentially enabling exploration of more fitness peaks as depicted in Figure 3B) at the expense of confidence that the variants will have non-zero fitness (Figure S6). While we previously used training sample sizes of 24 and 48 to test the effectiveness of different encodings in the low-N setting, here we include them to enable comparison of ftMLDE (and standard MLDE) with the most efficient implementation of traditional DE. As discussed previously in the encoding comparison section, traditional DE can in principle be performed on a four-site library by deterministically evaluating all 20 amino acids at each position, requiring only 80 measurements for the GB1 landscape. Again, due to the cost of synthesizing variants individually, this approach is rarely taken. However, use of 24- and 48-variant training sets (with 56 and 32 tested predictions, respectively) allows for direct comparison of the *algorithms* of ftMLDE and this most efficient implementation of traditional DE.

For each of the 66 training data types, simulated MLDE was performed using each training set with variants encoded using either one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer (which was the most effective of the learned embeddings tested earlier) (zero-shot simulations). In total, testing all encodings with all training data types amounted to 198 "training conditions" (66 training data types × 3 encodings/type) and 396,000 simulated MLDE experiments (198 training conditions × 2,000 simulations/condition = 396,000 simulated MLDE experiments). As before, cross-validation indices and random seeds were kept the same between simulations using different encodings but the same training data. For each simulation, after prediction, only the top-predicted unsampled combinations that could be constructed by recombining combinations in the training data were evaluated (e.g., if "AAAA" and "CCCC" were the only training examples, then only "AAAC," "AACC," "CAAA," etc. could be in the top M proteins chosen for fitness evaluation). This approach enforced a confidence threshold on our predictions and focused all resources on regions believed to contain the highest-fitness protein variants.

The distributions of the achieved max and mean fitness for all simulations with a training sample size of 384 are shown in Figures 5 and 6, respectively. Distributions of the achieved max and mean fitness for simulations with smaller training sample sizes of 24 and 48 are shown in Figures S7–S10 and summary statistics for all simulations are provided in Data S3. Both MLDE and ftMLDE using 384 training samples outperformed traditional DE regardless of encoding and zero-shot strategy, with the most effective set of simulations (ftMLDE run using training data sampled from the top-3,200 Triad predictions and the MSA Transformer for encoding) achieving the global maximum in 99.70% of simulations. By comparison, simulated traditional DE on the GB1 landscape reached the global optimum just 1.23% of the time (traditional directed evolution simulations). At lower screening burdens, both MLDE and ftMLDE remained competitive with traditional DE (in terms of mean- and median-maximum fitness achieved over all simulations), though only ftMLDE simulations ever achieved the global optimum more frequently than traditional DE. Specifically, ftMLDE simulations using 24 training samples achieved the GB1 global optimum more frequently than traditional DE in 40 out of 63 ftMLDE training conditions; ftMLDE simulations using 48 training samples achieved the GB1 global optimum more frequently than traditional DE in 57 out of 63 training conditions tested. Almost all training conditions where ftMLDE did not outcompete traditional DE in the low-sample setting used mask filling with the MSA Transformer as the zero-shot predictor, with 0 out of 21 such

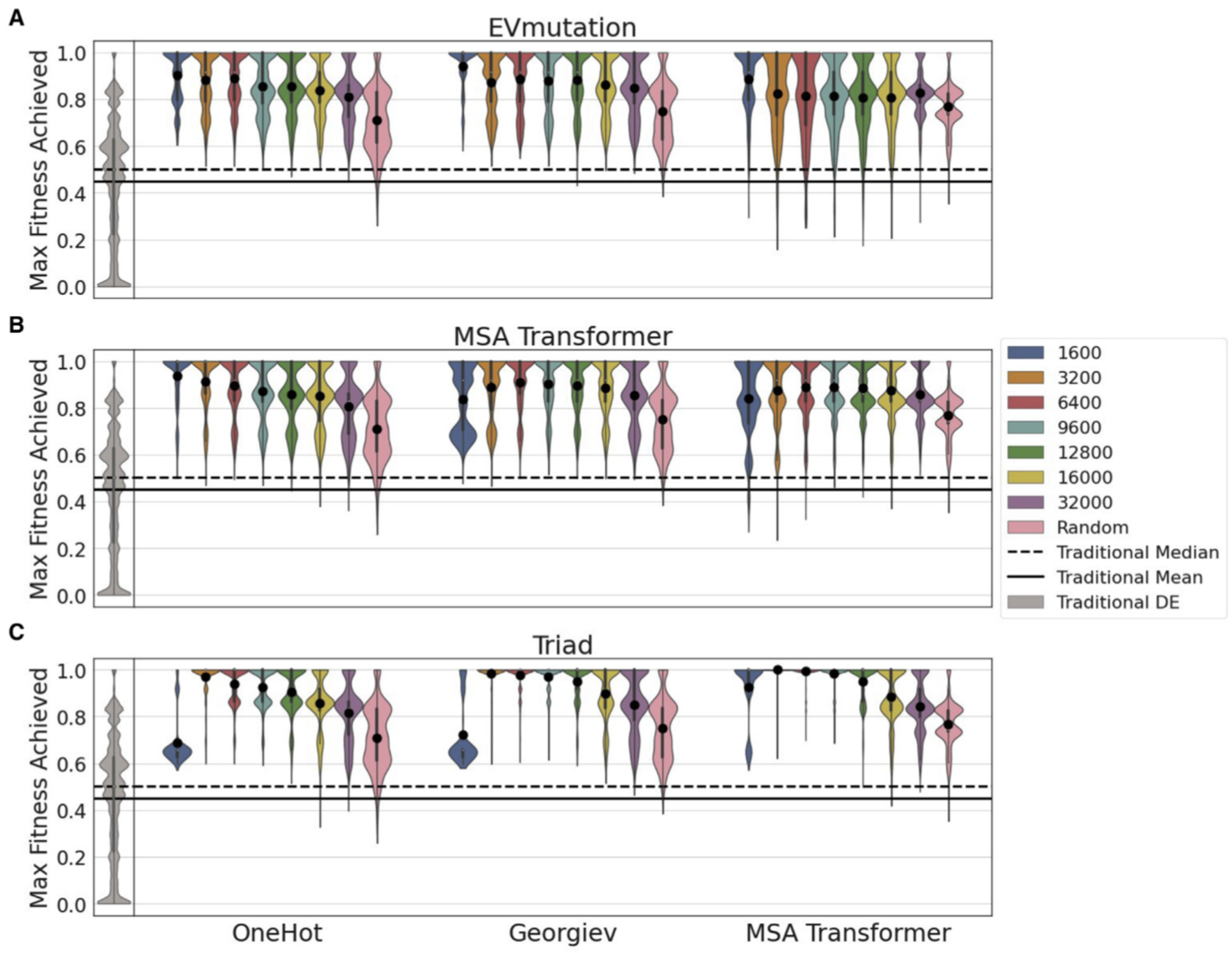

Figure 5. Zero-shot prediction for training set design enables highly effective ftMLDE on the GB1 landscape, as measured by maximum fitness achieved in simulated experiments

Each subplot (A–C) shows the effect of different zero-shot predictors on the maximum fitness achieved in simulated ftMLDE experiments. Each violin (except for the gray ones corresponding to simulated traditional DE) represents data from 2,000 simulated experiments where 384 variants were used for training and the top 96 predictions were tested. The major groupings of violins within each subplot correspond to different encoding strategies (one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer). The color of each violin corresponds to the zero-shot sampling threshold (i.e., the number of best-ranked variants according to a zero-shot predictor from which random samples were drawn to generate training data). Results of ftMLDE are compared with the results of simulated traditional DE (at the left of each plot, in gray) and standard MLDE (the three pink violins in each plot).

(A) The maximum fitness achieved by simulated ftMLDE when EVmutation was used as the zero-shot predictor for training set design.

(B) The maximum fitness achieved by simulated ftMLDE when a mask-filling protocol using the MSA Transformer was used as the zero-shot predictor for training set design.

(C) The maximum fitness achieved by simulated ftMLDE when predicted ΔΔG was used as the zero-shot predictor for training set design. See also, Figures S6–S8 and Data S3.

conditions outcompeting traditional DE at a training sample size of 24 and 15 out of 21 at a training sample size of 48. It is also notable that training on 48 samples and testing 32 tended to be a more effective strategy than training on 24 samples and testing 56, as this result indicates that, at least for GB1, devoting screening resources to the training stage of the MLDE workflow may be more important than the testing phase. Indeed, the most effective set of ftMLDE simulations at low screening burden was with 48 training samples (from the top-3200 Triad predictions and encoded using Georgiev parameters), where the global maximum was achieved 9.95% of the time.

Aside from comparisons to traditional DE, the results of our simulations allow direct comparison of ftMLDE and MLDE and show that ftMLDE is generally a more effective strategy for navigating the GB1 landscape than MLDE. The optimal MLDE strategy, for instance, achieved the global optimum just 8.85% of the time compared with the 99.70% of the optimal ftMLDE strategy. Additionally, in almost all training conditions tested, ftMLDE tended to achieve higher mean and max fitness than the comparable MLDE control. There are some exceptions, however, that suggest that the combination of training set diversity, zero-shot strategy, and encoding have

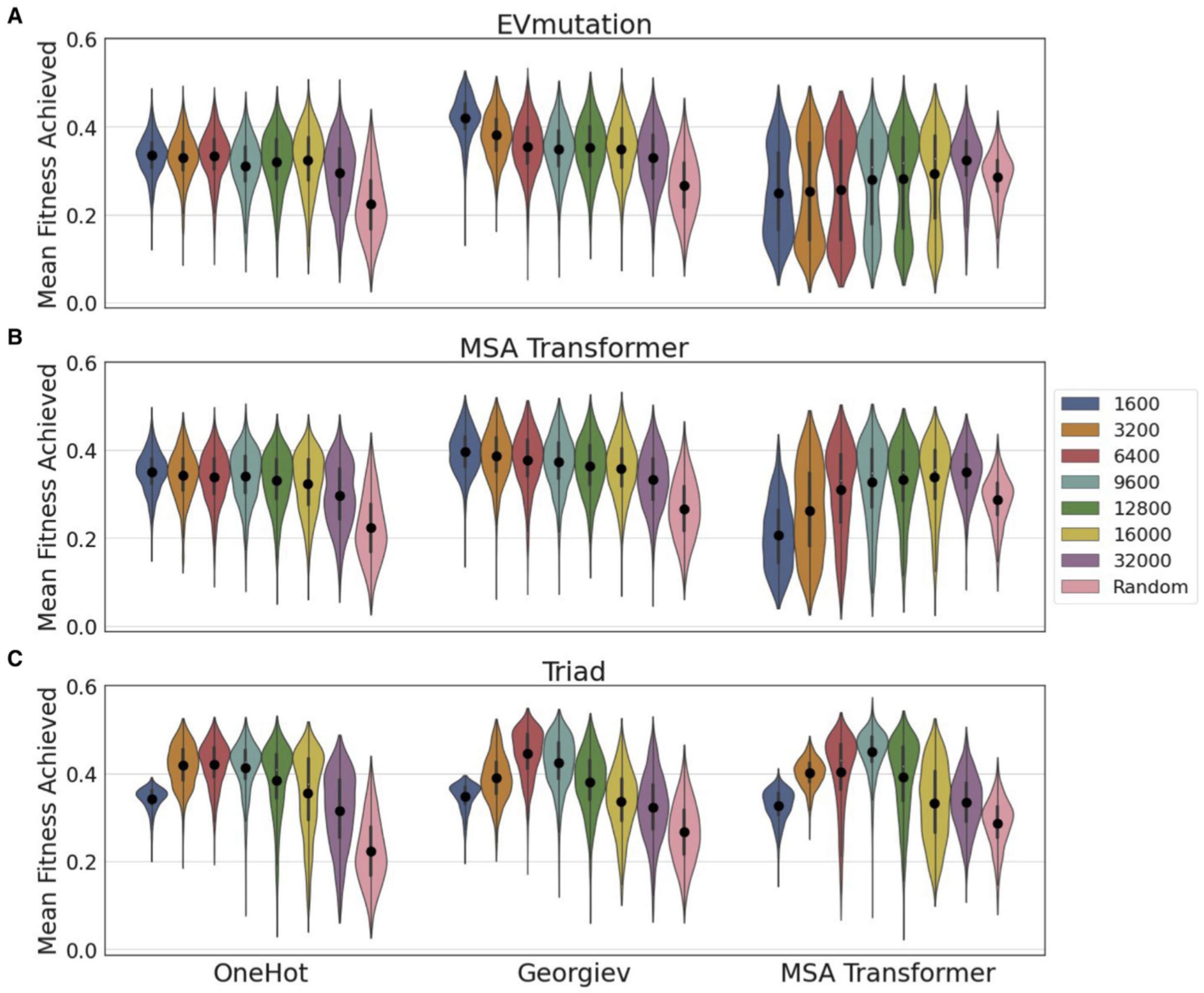

Figure 6. Zero-shot prediction for training set design enables highly effective ftMLDE on the GB1 landscape, as measured by mean fitness achieved in simulated experiments

Each subplot (A–C) shows the effect of different zero-shot predictors on the mean fitness achieved in simulated ftMLDE experiments. Each violin represents data from 2,000 simulated experiments where 384 variants were used for training and the top 96 predictions were tested. The major groupings of violins within each subplot correspond to different encoding strategies (one-hot, Georgiev parameters, or learned embeddings from the MSA Transformer). The color of each violin corresponds to the zero-shot sampling threshold (i.e., the number of best-ranked variants according to a zero-shot predictor from which random samples were drawn to generate training data). Results of ftMLDE are compared with the results of standard MLDE (the three pink violins in each plot).

(A) The mean fitness achieved by simulated ftMLDE when EVmutation was used as the zero-shot predictor for training set design.

(B) The mean fitness achieved by simulated ftMLDE when a mask-filling protocol using the MSA Transformer was used as the zero-shot predictor for training set design.

(C) The mean fitness achieved by simulated ftMLDE when predicted ΔΔG was used as the zero-shot predictor for training set design. See also, Figures S6, S9, and S10 and Data S3.

an effect on the outcome of ftMLDE. For instance, all ftMLDE training conditions tended to achieve a higher max fitness than the relevant MLDE control except those using 384 training points derived from the top-1,600 Triad samples and encoding with one-hot or Georgiev parameters. Similarly, ftMLDE tended to achieve a higher mean fitness than the relevant MLDE control in all training conditions tested except for a number using learned embeddings from the MSA Transformer for encoding with 384 training points derived from sequence-based zero-shot predictors (EVmutation and mask filling using the MSA Transformer).

The reasons for the observed exceptions to ftMLDE's general superiority over MLDE are not immediately clear, though it is interesting to note that (1) the only training condition run using 384 training points from the top-1,600 Triad samples that achieved higher max fitness than MLDE was the one using MSA Transformer encodings and (2) the training conditions where ftMLDE achieved a lower mean fitness than MLDE were those trained using data encoded with the MSA Transformer that was derived from sequence-based zero-shot predictors. During training, the embeddings of the MSA Transformer were developed to be able to predict the identity of masked amino acids (see a discussion of the masked-token training procedure above in leveraging sequence data for the design of fitness-enriched training data) (Rao et al., 2021). The embeddings themselves thus contain information about what mutations are and are not likely in a given reference protein based on available sequence data. Indeed, if they did not, we would be unable to successfully make zero-shot predictions using a mask-filling protocol with the MSA Transformer model. It is interesting to ask, then, if models trained on data derived from zero-shot predictions made by Triad and encoded using MSA Transformer embeddings have access to two sets of prior information (one derived from the data via physics-based Triad calculations and another from sequence conservation captured in the MSA Transformer embeddings), thus making them more effective than models trained with encodings that do not capture fitness information from sequence. Similarly, it could be asked if models trained on data derived from sequence-based zero-shot predictors and encoded by the MSA Transformer become overly restricted by a “double-dose” of sequence-based prior information. Such effects could explain both observations at the beginning of this paragraph, and, indeed, why the combination of Triad-derived data and the MSA Transformer was the most effective ftMLDE strategy tested. This is, of course, conjecture, and the only clear conclusion that we can derive from these results is that there is an interplay between training data makeup and encoding strategy in determining ftMLDE outcome.

MLDE software enables wet-lab application

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

DISCUSSION

We have demonstrated improvements to MLDE that, all together, can make it more efficient than the lowest-possible-screening-burden form of DE for navigating an epistatic, hole-filled, combinatorial protein fitness landscape. While incorporation of more informative encodings and models/regression strategies more amenable to combinatorial protein fitness landscapes was shown to improve MLDE outcome somewhat, by far the greatest improvement came from training set design. Specifically, we show that a focused training MLDE (ftMLDE) strategy that uses some type of predictor to avoid minimally informative extremely low-fitness variants in the training data is typically more capable than standard MLDE at identifying the most-fit variants in a combinatorial landscape. From simulated experiments, we note that the predictor used for training set design in ftMLDE does not need to be capable of identifying particularly high-fitness variants—in tests run using training data purposefully enriched in fitness, eliminating holes had a larger effect on outcome than subsequently raising training data mean fitness. The ability of the predictor to identify diverse sequences, however, is important for improving the probability of identifying the global maximum of a combinatorial landscape. This concept is best highlighted when using predicted ΔΔG of protein stability as a zero-shot strategy for building training sets, where a balance between sequence diversity and sequence fitness in the training data proved important for maximizing ftMLDE effectiveness (Figures 5C, 6C, and S6). It is also worth noting that there appears to be an interplay between zero-shot and encoding strategies used and ftMLDE effectiveness, with some combinations of zero-shot predictor and encoding strategy underperforming an MLDE control.

There is, of course, no guarantee that the zero-shot strategies found to be successful for GB1 would be effective for other proteins or other functions. The use of a sequence-based zero-shot strategy, for instance, assumes that the target fitness is well represented by evolutionarily optimized fitness, which will not be the case for all protein engineering problems. Likewise, use of a strategy like predicted ΔΔG assumes that stability (or, potentially, structural conservation) plays a role in fitness determination. In general, the optimal training set design strategy will depend on the protein (Livesey and Marsh, 2020), and while we have mainly discussed unsupervised zero-shot strategies (i.e., those working off protein sequence or structural data alone) in this work, alternate strategies can be imagined. For instance, if a protein scaffold has been used in previous protein engineering studies, a crude non-computational approach would be to avoid mutations that previously destroyed protein function. More robustly, a transfer learning approach could be taken, where an ML model trained using information from related experiments (e.g., evolution of the same protein for a different task, evolution of a different protein for the same task, or even data from previous rounds of MLDE at different positions) is used to predict the effects of mutations in the present experiment (Shamsi et al., 2020). Perhaps even more effectively, fitness information from single-site saturation mutagenesis or error-prone PCR random mutagenesis libraries could be used to predict the fitness of combinations. Indeed, Biswas et al., Hie et al., and Hsu et al. each recently demonstrated approaches where ML models trained on single-site or random mutation data were capable of predicting the fitness of combinations of those mutations (Biswas et al., 2021; Hie et al., 2020; Hsu et al., 2021). The use of Gaussian processes in the application of Hie et al. is particularly interesting, as it enables use of the upper confidence bound algorithm to explicitly balance exploration and exploitation, thus providing a more principled way to inject sequence diversity into training set design while maintaining high fitness (Hie et al., 2020; Srinivas et al., 2010).

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

Thorough evaluation of the effectiveness of ftMLDE will only be possible once more combinatorial landscape data beyond that provided by the GB1 landscape become available. For now, however, ftMLDE can be used on combinatorial landscapes known to be highly epistatic and that either contain few holes or else for which confident training set design strategies can be employed. The strategies, concepts, and technology presented in this work will serve as a foundation for further evaluation of the generalizability of different encodings, model architectures, regression strategies, and training set design strategies for ftMLDE on combinatorial fitness landscapes. By achieving the GB1 global maximum up to 99.70% of the time with a total screening burden of 480 protein variants, or up to 9.95% of the time with a screening burden of just 80 variants, the ftMLDE protocol presented here outcompeted both traditional DE—which achieved the global optimum just 1.23% of the time—and our original implementation—which achieved the global optimum 8.17% of the time with a screening burden of 570 variants (Wu et al., 2019). This work thus presents a large advance and is, to the best of our knowledge, the first proven example of an ML approach directly outcompeting minimalistic DE. Given the degree to which ftMLDE outcompetes traditional DE on the GB1 landscape, we hope for many more examples to come.

STAR★METHODS

Detailed methods are provided in the online version of this paper and include the following:

- KEY RESOURCES TABLE
- RESOURCE AVAILABILITY
  - Lead contact
  - Materials availability
  - Data and code availability
- METHOD DETAILS
  - Alignment generation and EVmutation model training
  - Encoding preparation
  - Zero-shot predictions
  - Mask-filling protocol
  - ΔΔG calculations
  - Simulation details
  - High-fitness simulations
  - Zero-shot simulations
  - Traditional directed evolution simulations
- QUANTIFICATION AND STATISTICAL ANALYSIS
  - Evaluation metrics
  - MLDE programmatic implementation
  - Inbuilt models
  - Compute environment
  - Computational hardware information

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.cels.2021.07.008.

ACKNOWLEDGMENTS

The authors thank Sabine Brinkmann-Chen, Patrick Almhjell, and Lucas Schaus for helpful discussion and critical reading of the manuscript, Zachary Wu, Kadina Johnston, and Amir Motmaen for helpful discussion, Suresh Guptha for assistance with computational infrastructure development and maintenance, and Paul Chang for assistance with Triad calculations. Additionally, the authors thank NVIDIA Corporation for donation of two Titan V GPUs used in this work and Amazon.com for donation of Amazon web services (AWS) computing credits. This work was supported by the NSF Division of Chemical, Bioengineering, Environmental and Transport Systems (CBET 1937902) and by an Amgen Chem-Bio-Engineering Award (CBEA).

AUTHOR CONTRIBUTIONS

Conceptualization, B.J.W., Y.Y., and F.H.A.; methodology, B.J.W. and Y.Y.; software, B.J.W.; validation, B.J.W.; formal analysis, B.J.W.; investigation, B.J.W.; writing – original draft, B.J.W., Y.Y., and F.H.A.; writing – review & editing, B.J.W., Y.Y., and F.H.A.; visualization, B.J.W.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: December 15, 2020
Revised: May 6, 2021
Accepted: July 26, 2021
Published: August 19, 2021

REFERENCES

Alley, E.C., Khimulya, G., Biswas, S., AlQuraishi, M., and Church, G.M. (2019). Unified rational protein engineering with sequence-based deep representation learning. Nat. Methods *16*, 1315–1322. https://doi.org/10.1038/s41592-019-0598-1.

Arnold, F.H. (2011). The library of Maynard-smith: my search for meaning in the protein universe. Microbe Magazine *6*, 316–318. https://doi.org/10.1128/microbe.6.316.1.

Arnold, F.H. (2018). Directed evolution: bringing new chemistry to life. Angew. Chem. Int. Ed. Engl. *57*, 4143–4148. https://doi.org/10.1002/anie.201708408.

Bai, S., Kolter, J.Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv, arXiv:1803.01271v2.

Bepler, T., and Berger, B. (2019). Learning protein sequence embeddings using information from structure. arXiv, arXiv:1902.08661.

Biswas, S., Khimulya, G., Alley, E.C., Esvelt, K.M., and Church, G.M. (2021). Low-N protein engineering with data-efficient deep learning. Nat. Methods *18*, 389–396. https://doi.org/10.1038/s41592-021-01100-y.

Bloom, J.D., Silberg, J.J., Wilke, C.O., Drummond, D.A., Adami, C., and Arnold, F.H. (2005). Thermodynamic prediction of protein neutrality. Proc.

Natl. Acad. Sci. USA *102*, 606–611. https://doi.org/10.1073/pnas.0406744102.

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. arXiv, arXiv:2005.14165.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., et al. (2013). API design for machine learning software: experiences from the scikit-learn project. arXiv, arXiv:1309.0238v1.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. arXiv, arXiv:1810.04805v2.

El-Gebali, S., Mistry, J., Bateman, A., Eddy, S.R., Luciani, A., Potter, S.C., Qureshi, M., Richardson, L.J., Salazar, G.A., Smart, A., et al. (2019). The Pfam protein families database in 2019. Nucleic Acids Res *47*, D427–D432. https://doi.org/10.1093/nar/gky995.

Elnaggar, A., Heinzinger, M., Dallago, C., Rihawi, G., Wang, Y., Jones, L., Feher, T., Angerer, C., Bhowmik, D., and Rost, B. (2020). ProtTrans: towards cracking the language of life's code through self-supervised deep learning and high performance computing. bioRxiv. https://doi.org/10.1101/2020.07.12.199554.

Firnberg, E., Labonte, J.W., Gray, J.J., and Ostermeier, M. (2014). A comprehensive, high-resolution map of a gene's fitness landscape. Mol. Biol. Evol. *31*, 1581–1592. https://doi.org/10.1093/molbev/msu081.

Franks, W.T., Wylie, B.J., Stellfox, S.A., and Rienstra, C.M. (2006). Backbone conformational constraints in a microcrystalline U-15N-labeled protein by 3d dipolar-shift solid-state nmr spectroscopy. J. Am. Chem. Soc. *128*, 3154–3155. https://doi.org/10.1021/ja058292x.

Georgiev, A.G. (2009). Interpretable numerical descriptors of amino acid space. J. Comput. Biol. *16*, 703–723. https://doi.org/10.1089/cmb.2008.0173.

Henikoff, S., and Henikoff, J.G. (1992). Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. USA *89*, 10915–10919. https://doi.org/10.1073/pnas.89.22.10915.

Hie, B., Bryson, B.D., and Berger, B. (2020). Leveraging uncertainty in machine learning accelerates biological discovery and design. Cell Syst *11*, 461–477.e9. https://doi.org/10.1016/j.cels.2020.09.007.

Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. Neural Comput *9*, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Hopf, T.A., Green, A.G., Schubert, B., Mersmann, S., Schärfe, C.P.I., Ingraham, J.B., Toth-Petroczy, A., Brock, K., Riesselman, A.J., Palmedo, P., et al. (2019). The EVcouplings Python framework for coevolutionary sequence analysis. Bioinformatics *35*, 1582–1584. https://doi.org/10.1093/bioinformatics/bty862.

Hopf, T.A., Ingraham, J.B., Poelwijk, F.J., Schärfe, C.P.I., Springer, M., Sander, C., and Marks, D.S. (2017). Mutation effects predicted from sequence co-variation. Nat. Biotechnol. *35*, 128–135. https://doi.org/10.1038/nbt.3769.

Hsu, C., Nisonoff, H., Fannjiang, C., and Listgarten, J. (2021). Combining evolutionary and assay-labelled data for protein fitness prediction. bioRxiv. https://doi.org/10.1101/2021.03.28.437402.

Iuchi, H., Matsutani, T., Yamada, K., Iwano, N., Sumi, S., Hosoda, S., Zhao, S., Fukunaga, T., and Hamada, M. (2021). Representation learning applications in biological sequence analysis. bioRxiv. https://doi.org/10.1101/2021.02.26.433129.

Jacquier, H., Birgy, A., Le Nagard, H., Mechulam, Y., Schmitt, E., Glodt, J., Bercot, B., Petit, E., Poulain, J., Barnaud, G., et al. (2013). Capturing the mutational landscape of the beta-lactamase TEM-1. Proc. Natl. Acad. Sci. USA *110*, 13067–13072. https://doi.org/10.1073/pnas.1215206110.

Jaganathan, K., Kyriazopoulou Panagiotopoulou, S.K., McRae, J.F., Darbandi, S.F., Knowles, D., Li, Y.I., Kosmicki, J.A., Arbelaez, J., Cui, W., Schwartz, G.B., et al. (2019). Predicting splicing from primary sequence with deep learning. Cell *176*, 535–548.e24. https://doi.org/10.1016/j.cell.2018.12.015.

Järvelin, K., and Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. *20*, 422–446. https://doi.org/10.1145/582415.582418.

Jiang, S., and Zavala, V.M. (2021). Convolutional neural nets: foundations, computations, and new applications. arXiv, arXiv:2101.0486.

Kawashima, S., Pokarowski, P., Pokarowska, M., Kolinski, A., Katayama, T., and Kanehisa, M. (2008). AAindex: amino acid index database, progress report 2008. Nucleic Acids Res *36*, D202–D205. https://doi.org/10.1093/nar/gkm998.

Kaznatcheev, A. (2019). Computational complexity as an ultimate constraint on evolution. Genetics *212*, 245–265. https://doi.org/10.1534/genetics.119.302000.

Li, G., Dong, Y., and Reetz, M.T. (2019). Can machine learning revolutionize directed evolution of selective enzymes? Adv. Synth. Catal. *361*, 2377–2386. https://doi.org/10.1002/adsc.201900149.

Li, M.M., Huang, K., and Zitnik, M. (2021). Representation learning for networks in biology and medicine: advancements, challenges, and opportunities. arXiv, arXiv:2104.04883v1.

Livesey, B.J., and Marsh, J.A. (2020). Using deep mutational scanning to benchmark variant effect predictors and identify disease mutations. Mol. Syst. Biol. *16*, e9380. https://doi.org/10.15252/msb.20199380.

Madani, A., McCann, B., Naik, N., Keskar, N.S., Anand, N., Eguchi, R.R., Huang, P.-S., and Socher, R. (2020). ProGen: language modeling for protein generation. bioRxiv. https://doi.org/10.1101/2020.03.07.982272.

Mazurenko, S., Prokop, Z., and Damborsky, J. (2020). Machine learning in enzyme engineering. ACS Catal *10*, 1210–1223. https://doi.org/10.1021/acscatal.9b04321.

Miton, C.M., and Tokuriki, N. (2016). How mutational epistasis impairs predictability in protein evolution and design. Protein Sci *25*, 1260–1272. https://doi.org/10.1002/pro.2876.

Ng, P.C., and Henikoff, S. (2003). SIFT: predicting amino acid changes that affect protein function. Nucleic Acids Res *31*, 3812–3814. https://doi.org/10.1093/nar/gkg509.

Nisthal, A., Wang, C.Y., Ary, M.L., and Mayo, S.L. (2019). Protein stability engineering insights revealed by domain-wide comprehensive mutagenesis. Proc. Natl. Acad. Sci. USA *116*, 16367–16377. https://doi.org/10.1073/pnas.1903888116.

Ofer, D., and Linial, M. (2015). ProFET: feature engineering captures high-level protein functions. Bioinformatics *31*, 3429–3436. https://doi.org/10.1093/bioinformatics/btv345.

Olson, C.A., Wu, N.C., and Sun, R. (2014). A comprehensive biophysical description of pairwise epistasis throughout an entire protein domain. Curr. Biol. *24*, 2643–2651. https://doi.org/10.1016/j.cub.2014.09.072.

Proutski, V., and Holmes, E. (1998). SWAN: sliding window analysis of nucleotide sequence variability. Bioinformatics *14*, 467–468. https://doi.org/10.1093/bioinformatics/14.5.467.

Rao, R., Bhattacharya, N., Thomas, N., Duan, Y., Chen, X., Canny, J., Abbeel, P., and Song, Y.S. (2019). Evaluating protein transfer learning with TAPE. Adv. Neural Inf. Process. Syst. 32, 9689–9701.

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

Rawat, W., and Wang, Z. (2017). Deep convolutional neural networks for image classification: a comprehensive review. Neural Comput *29*, 2352–2449. https://doi.org/10.1162/NECO_a_00990.

Riesselman, A., Shin, J.-E., Kollasch, A., McMahon, C., Simon, E., Sander, C., Manglik, A., Kruse, A., and Marks, D. (2019). Accelerating protein design using autoregressive generative models. bioRxiv. https://doi.org/10.1101/757252.

Riesselman, A.J., Ingraham, J.B., and Marks, D.S. (2018). Deep generative models of genetic variation capture the effects of mutations. Nat. Methods *15*, 816–822. https://doi.org/10.1038/s41592-018-0138-4.

Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C.L., Ma, J., and Fergus, R. (2021). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. Proc. Natl. Acad. Sci. USA *118*. https://doi.org/10.1073/pnas.2016239118.

Romero, P.A., and Arnold, F.H. (2009). Exploring protein fitness landscapes by directed evolution. Nat. Rev. Mol. Cell Biol. *10*, 866–876. https://doi.org/10.1038/nrm2805.

Romero, P.A., Krause, A., and Arnold, F.H. (2013). Navigating the protein fitness landscape with Gaussian processes. Proc. Natl. Acad. Sci. USA *110*, E193–E201. https://doi.org/10.1073/pnas.1215251110.

Sarkisyan, K.S., Bolotin, D.A., Meer, M.V., Usmanova, D.R., Mishin, A.S., Sharonov, G.V., Ivankov, D.N., Bozhanova, N.G., Baranov, M.S., Soylemez, O., et al. (2016). Local fitness landscape of the green fluorescent protein. Nature *533*, 397–401. https://doi.org/10.1038/nature17995.

Shamsi, Z., Chan, M., and Shukla, D. (2020). TLmutation: predicting the effects of mutations using transfer learning. J. Phys. Chem. B *124*, 3845–3854. https://doi.org/10.1021/acs.jpcb.0c00197.

Shanehsazzadeh, A., Belanger, D., and Dohan, D. (2020). Is transfer learning necessary for protein landscape prediction? arXiv, arXiv:2011.03443v1.

Siedhoff, N.E., Schwaneberg, U., and Davari, M.D. (2020). Machine learning-assisted enzyme engineering. Methods Enzymol *643*, 281–315. https://doi.org/10.1016/bs.mie.2020.05.005.

Sinai, S., and Kelsic, E. (2020). A primer on model-guided exploration of fitness landscapes for biological sequence design. arXiv, arXiv:2010.10614.

Sirin, S., Apgar, J.R., Bennett, E.M., and Keating, A.E. (2016). AB-bind: antibody binding mutational database for computational affinity predictions. Protein Sci *25*, 393–409. https://doi.org/10.1002/pro.2829.

Smith, J.M. (1970). Natural selection and the concept of a protein space. Nature *225*, 563–564. https://doi.org/10.1038/225563a0.

Srinivas, N., Krause, A., Kakade, S.M., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: no regret and experimental design. arXiv, arXiv:0912.3995.

Starr, T.N., and Thornton, J.W. (2016). Epistasis in protein evolution. Protein Sci *25*, 1204–1218. https://doi.org/10.1002/pro.2897.

Steinegger, M., and Söding, J. (2018). Clustering huge protein sequence sets in linear time. Nat. Commun. *9*, 2542. https://doi.org/10.1038/s41467-018-04964-5.

Tajima, F. (1991). Determination of window size for analyzing DNA sequences. J. Mol. Evol. *33*, 470–473. https://doi.org/10.1007/BF02103140.

Chen, T., and Guestrin, C. (2016). XGBoost: a scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ACM), pp. 785–794. https://doi.org/10.1145/2939672.2939785.

UniProt Consortium. (2019). UniProt: a worldwide hub of protein knowledge. Nucleic Acids Res *47*, D506–D515. https://doi.org/10.1093/nar/gky1049.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. arXiv, arXiv:1706.03762.

Vig, J., Madani, A., Varshney, L.R., Xiong, C., Socher, R., and Rajani, N.F. (2020). BERTology meets biology: interpreting attention in protein language models. arXiv, arXiv:2006.15222.

Wittmann, B.J., Johnston, K.E., Wu, Z., and Arnold, F.H. (2021). Advances in machine learning for directed evolution. Curr. Opin. Struct. Biol. *69*, 11–18. https://doi.org/10.1016/j.sbi.2021.01.008.

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

Wu, Z., Kan, S.B.J., Lewis, R.D., Wittmann, B.J., and Arnold, F.H. (2019). Machine learning-assisted directed protein evolution with combinatorial libraries. Proc. Natl. Acad. Sci. USA *116*, 8852–8858. https://doi.org/10.1073/pnas.1901979116.

Xu, Y., Verma, D., Sheridan, R.P., Liaw, A., Ma, J., Marshall, N.M., McIntosh, J., Sherer, E.C., Svetnik, V., and Johnston, J.M. (2020). Deep dive into machine learning models for protein engineering. J. Chem. Inf. Model. *60*, 2773–2790. https://doi.org/10.1021/acs.jcim.0c00073.

Yang, J., Naik, N., Patel, J.S., Wylie, C.S., Gu, W., Huang, J., Ytreberg, F.M., Naik, M.T., Weinreich, D.M., and Rubenstein, B.M. (2020). Predicting the viability of beta-lactamase: how folding and binding free energies correlate with beta-lactamase fitness. PLoS One *15*, e0233509. https://doi.org/10.1371/journal.pone.0233509.

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

Yang, Y., Qian, W., and Zou, H. (2018). Insurance premium prediction via gradient tree-boosted tweedie compound poisson models. J. Bus. Econ. Stat. *36*, 456–470. https://doi.org/10.1080/07350015.2016.1200981.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. IEEE Comput. Intell. Mag. *13*, 55–75. https://doi.org/10.1109/MCI.2018.2840738.

Yu, F., Koltun, V., and Funkhouser, T. (2017). Dilated residual networks. arXiv, arXiv:1705.09914.

Zhang, Y., Zhou, X., and Cai, X. (2020). Predicting gene expression from DNA sequence using residual neural network. bioRxiv. https://doi.org/10.1101/2020.06.21.163956.

Zhou, H., Qian, W., and Yang, Y. (2020). Tweedie gradient boosting for extremely unbalanced zero-inflated data. Commun. Stat. Simul. Comput. 1–23. https://doi.org/10.1080/03610918.2020.1772302.

Zhu, Z., Wang, Y., Zhou, X., Yang, L., Meng, G., and Zhang, Z. (2020). SWAV: a web-based visualization browser for sliding window analysis. Sci. Rep. *10*, 149. https://doi.org/10.1038/s41598-019-57038-x.

STAR★METHODS

KEY RESOURCES TABLE

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

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Frances Arnold (frances@cheme.caltech.edu).

Materials availability

This study did not generate new unique reagents.

Data and code availability

- Data needed to replicate simulations have been deposited at Caltech Data and are publicly available as of the date of publication. DOIs are listed in the key resources table. The raw simulation data reported in this study cannot be deposited in a public repository because it is multiple terabytes in size. To request access, contact Bruce Wittmann at bwittman@caltech.edu. In addition, summary statistics describing these raw data have been deposited at Caltech Data and are publicly available as of the date of publication. DOIs are listed in the key resources table. This paper analyzes existing, publicly available data. The accession numbers for the datasets are listed in the key resources table.
- All original code has been deposited at Caltech Data and is publicly available as of the date of publication. DOIs are listed in the key resources table.
- Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

METHOD DETAILS

Alignment generation and EVmutation model training

The EVcouplings webapp (Hopf et al., 2019) was used to both generate multiple sequence alignments (MSAs) as well as train the EVmutation model needed for zero-shot prediction. The GB1 sequence (See encoding preparation, below) was passed into the EVcouplings webapp, and alignments were made against the UniRef100 database for bitscore inclusion thresholds of 0.30, 0.35, 0.40, 0.45, and 0.50. All other EVcouplings parameters were kept at their default values (Alignment threshold type = Bitscore; Search iterations = 5; Position filter = 70%; Sequence fragment filter = 50%; Removing similar sequences = 90%; Downweighting similar sequences = 80%). A bitscore threshold of 0.40 returned an alignment with the most redundancy-reduced sequences (56) that covered all variable positions in the GB1 landscape at $\geq$70% coverage (i.e., less than 30% of aligned sequences had gaps at the positions of interest). Bitscores of 0.30 and 0.35 returned more redundancy-reduced sequences (2427 and 664, respectively), but failed to cover position 54 in the landscape. Bitscores of 0.45 and 0.50 covered all positions, but returned less redundancy-reduced sequences (30 and 27, respectively). We decided to move forward with the alignment and EVmutation model generated at a bitscore of 0.40. The alignment (in.a2m format) and the parameters for the EVmutation model trained on it (the ".model" file) were downloaded from the EVcouplings webapp. The alignment downloaded would also be used to train a DeepSequence VAE as well as build encodings and make zero-shot predictions using the MSA Transformer (See encoding preparation, EVmutation/DeepSequence calculations, and mask-filling protocol, all below).

Encoding preparation

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

Physicochemical encodings were prepared using the descriptors originally published by Georgiev (Georgiev, 2009), using the values found in code published by Ofer & Linial (Ofer and Linial, 2015). To encode all variants, a 160,000×4×19 tensor was instantiated ("N possible combos" × "N amino acids per combo" × "N Georgiev parameters"). Every possible variant was encoded using all 19 Georgiev parameters, and the encodings were stored in the instantiated tensor. The last two dimensions of the tensor were then flattened to produce a 160,000×76 matrix and each column of the matrix was mean-centered and unit-scaled. The final encoding tensor was generated by extracting only those rows belonging to GB1 variants with experimentally measured fitness, then reshaping the last dimension to produce a 149,361×4×19 tensor.

Learned embeddings were prepared using the pre-trained models published in the TAPE, ESM, and ProtTrans GitHub repositories (Alley et al., 2019; Bepler and Berger, 2019; Elnaggar et al., 2020; Rao et al., 2019, 2021; Rives et al., 2021). For the non-MSA Transformer models, embeddings were generated by first building fasta files containing the full amino acid sequences of all 160,000 possible GB1 variants at positions 39, 40, 41, and 54. The template sequence used was:

MQYKLILNGKTLKGETTTEAVDAATAEKVFKQYANDNGVDGEWTYDDATKTFTVTE

The sequences contained in the fasta file were then embedded using the appropriate code provided by each of the repositories. For the TAPE models ("Bepler", "ResNet", "UniRep", "TAPE-Transformer", and "LSTM") the tape-embed command from the software associated with the original publication by Rao et al. (https://github.com/songlab-cal/tape-neurips2019) was used to generate embeddings. For the ESM model (esm1b_t33_650M_UR50S), the example code provided in the "Quick Start" section of the GitHub repository (https://github.com/facebookresearch/esm#quick-start-) was used to generate embeddings. For the ProtTrans model ("ProtBert-BFD"), code from the example Jupyter notebook provided on the associated GitHub repository (https://github.com/agemagician/ProtTrans/blob/master/Embedding/PyTorch/Advanced/ProtBert-BFD.ipynb) was used to generate embeddings. Each of these processes generated tensors of shape 160,000×S×L ("N possible combos" × "Tokenized Sequence Length" × "L latent dimensions"). The value of L varied by encoding used, and is given in Table S1. The tokenized sequence length (S) varied depending on whether or not the embeddings for "<cls>" and "<eos>" tokens were returned alongside the embeddings for the different amino acids. The value of S was taken into account in the next step, where the embeddings corresponding to amino acids varied in the GB1 dataset (Using 0-indexing, this was indices 38, 39, 40, and 53 of the middle dimension of the output tensor if no <cls> token was added, and 39, 40, 41, and 54 if a <cls> token was added) were extracted from the output tensor. Using the same procedure as with the physicochemical parameters, the resultant 160,000×4×L tensor was mean-centered and unit-scaled to produce a 160,000×4L matrix. Finally, the appropriate rows were isolated and the last dimension reshaped to produce a final learned embedding encoding tensor of shape 149,361×4×L.

Unlike all other models tested, the MSA Transformer takes an MSA as an input rather than a sequence. To build variant MSAs, we used the MSA generated by the EVcouplings webserver (See alignment generation and EVmutation model training, above) as a template. To begin, we filtered out all lowercase characters and the "." character for each sequence in the template MSA. Next, we removed any duplicate sequences from the filtered MSA. We used this filtered and de-duplicated MSA as a template to build an

MSA for each mutant in the GB1 landscape. When building mutants, we changed only the GB1 reference sequence, keeping all other sequences in the MSA constant. The resultant 160,000-mutant MSAs were then streamed through the MSA Transformer, extracting the embeddings for the mutant GB1 positions in the first sequence of each alignment (corresponding to the mutant GB1 sequence). This procedure resulted in a 160,000×4×768 tensor (where "768" is the number of latent dimensions assigned to each token by the MSA Transformer). This tensor was then mean-centered and unit-scaled following the same procedure as for all other encodings before being filtered to produce a tensor of shape 149,361×4×768.

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

Zero-shot predictions

EVmutation/DeepSequence calculations

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

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

Mask-filling protocol

Zero-shot predictions using a mask-filling protocol were performed for all models provided in the ESM GitHub repository as well as the models ProtBert and ProtBert-BFD in the ProtTrans GitHub repository (Elnaggar et al., 2020; Rao et al., 2021; Rives et al., 2021). For each of these models, we tested both a naïve and conditional mask-filling strategy for making zero-shot predictions of GB1 fitness. For GB1, both protocols model the probability

$$P(combo|s_{const}) = P(aa_{39}aa_{40}aa_{41}aa_{54}|s_{const}), \qquad \text{(Equation 2)}$$

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

$$P(combo|s_{const}) = P(aa_{39}|s_{const})P(aa_{40}|s_{const})P(aa_{41}|s_{const})P(aa_{54}|s_{const}). \qquad \text{(Equation 3)}$$

Conditional probability, in contrast, does not assume independence of the variable positions and instead directly solves the probability given by Equation 2 using the product rule of probability. Note that, for all non-MSA Transformer methods, the parent GB1 sequence was used to define s_{const}, while for the MSA Transformer, the MSA used for EVmutation (after applying the same filtering and de-duplication procedure from encoding preparation, above) was used to define s_{const}. Only the variable positions in the reference sequence of this MSA were masked (rather than the full alignment column corresponding to variable positions), so the MSA Transformer maintained full access to the information provided by other sequences in the MSA.

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

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

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

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

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

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

Calculation of the conditional probability was much more expensive than the calculation of naïve probability. For instance, while determination of naïve probability for GB1 required calculating just the variable-position logits of the GB1 sequence with variable positions masked (a single pass through a model), calculation of conditional probability required calculating the variable-position logits of all 34,481 possible masked combinations. While trivial for smaller models, calculating conditional probability using larger models with complex transformations (such as the MSA Transformer or ESM1b) could become expensive. Regardless, calculation of component probabilities used for calculating the overall conditional probably was performed in the same way as for naïve probability. For each component probability: (1) appropriate positions in the GB1 sequence were masked, (2) a softmax was taken over the alphabet dimension, and (3) the appropriate probabilities were extracted from the resultant probability matrix. Note that, unlike for naïve calculation, we did not take the log of the calculated probability matrices—the necessity of calculating a sum over the different factorizations precluded this possibility. While such a practice may lead to numerical instability for large combinatorial libraries, we observed no such problems for GB1.

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

ΔΔG calculations

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

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

This command generated two files: 2GI9_process.pdb and 2GI9_prepared.pdb (Franks et al., 2006). The "_process" pdb file is the 2GI9.pdb file prepared for downstream Triad calculations but without any structural minimization. The "_prepared" pdb file is the 2GI9.pdb file prepared for downstream Triad calculations but with an added constrained minimization. The flexible backbone calculations were run using the standard Rosetta scoring function and the "_prepared" pdb file. The command line call is below:

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

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

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

There was no output file directly produced by the cleanSequence.py script, hence the captured output. The captured output file generated was parsed to extract ΔG values for each protein variant, which were in turn used to calculate ΔΔG values relative to the parent protein. In this work, we defined a negative ΔΔG to be stabilizing and a positive ΔΔG to be destabilizing relative to the parent protein; this necessitated flipping the sign of the literature ΔΔG values when building Figure S2, as Nisthal et al. defined opposite meanings of the sign of ΔΔG (Nisthal et al., 2019).

Simulation details

Encoding comparison simulations

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

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

High-fitness simulations

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

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

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

Zero-shot simulations

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

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

For the 24- and 48-training-sample simulations, the first 24 and 48 variants in each of the full 384-sample training sets were used for training, respectively. The models ARDRegression, BaggingRegressor, and KNeighborsRegressor were omitted from the ensemble trained when using the MSA Transformer for variant encoding. Otherwise, the procedure was the same as for the 384-training-sample simulations.

Traditional directed evolution simulations

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

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

QUANTIFICATION AND STATISTICAL ANALYSIS

Evaluation metrics

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

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

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

MLDE programmatic implementation

The MLDE algorithm takes as input all encodings corresponding to the combinations of amino acids found in the training data along with their measured fitness values. During the training stage, these sampled combinations are used to train a version of all inbuilt model architectures (inbuilt models). Specifically, k-fold cross-validation (Box 1) is performed to train each model using the default model parameters; mean validation error (Box 1) from the k-fold cross-validation (mean squared error) is recorded for each architecture. All model instances trained during k-fold cross-validation are also stored for later use. For instance, if evaluating all 22 inbuilt model architectures with 5-fold cross-validation, 22 × 5 = 110 total *trained* model instances are recorded. For making predictions, the top N model architectures (those with the lowest cross-validation error) are first identified. For each of the top N model architectures, predictions are made on the unsampled combinations by averaging the predictions of the k×N model instances stored during cross-validation. For instance, if testing the top 3 model architectures identified from 5-fold cross-validation, this means that the predictions of 3 × 5 = 15 total models (3 architectures × 5 model instances/architecture saved during cross-validation) are used for prediction. For all simulations presented in this work, we evaluated model architectures using 5-fold cross-validation and then made predictions using the top 3 (again, meaning that the predictions of 3 × 5 = 15 total models were averaged for each simulation).

Inbuilt models

Keras

Five separate neural network architectures were implemented using the Python package Keras: three fully connected neural network architectures and two 1D-convolutional neural network architectures. The identities and default values of tunable hyperparameters are given in Table S8. All neural networks were trained with a batch size of 32 using the "adam" optimizer for at most 1000 epochs with early stopping after 10 epochs with no improvement in validation error (calculated against the cross-validation test data).

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

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

XGBoost

Four gradient boosting approaches were implemented in MLDE using the Python package XGBoost (Chen and Guestrin, 2016): both tree and linear base models were implemented with both "reg:squarederror" and "reg:tweedie" objectives. For reg:tweedie, 'tweedie_variance_power' was set to 1.5. The identities and default values for tunable XGBoost hyperparameters are given in Table S9. Unless explicitly mentioned in Table S9, all XGBoost parameters were held at their default values as detailed in the official XGBoost documentation. The descriptions for all parameters can also be found in the official XGBoost documentation. All XGBoost models used in this work were implemented with early stopping: 'eval_metric' was set to "rmse" when the "reg:squarederror" was used and "tweedie-nloglik@1.5" when the "reg:tweedie" objective was used; validation error was calculated against the cross-validation test data; training was terminated if validation error did not decrease for 10 epochs or 1000 total training epochs had passed.

Scikit-learn

Only scikit-learn models were used in our original implementation of MLDE (Buitinck et al., 2013; Wu et al., 2019). To remain consistent with this first implementation, the regressor models from scikit-learn that were procedurally effective (i.e., those that did not consistently error during operation or else take a very long time to train) while using default parameters in our previous implementation were also used in this version. The identities and default values for tunable scikit-learn hyperparameters are given in Table S10. Unless explicitly mentioned in Table S10, all scikit-learn parameters were held at their default values as detailed in the official scikit-learn documentation. The descriptions for all parameters can also be found in the official documentation.

Compute environment

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

Computational hardware information

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