Supporting Information for

Machine Learning-Assisted Protein Evolution with Combinatorial Libraries

Zachary Wu[a], S. B. Jennifer Kan[a], Russell D. Lewis[b], Bruce J. Wittmann[b], Frances H. Arnold[a,b,1]

[a]Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, CA 91125; and [b]Division of Biology and Bioengineering, California Institute of Technology, Pasadena, CA 91125

[1]To whom correspondence may be addressed. Email: frances@cheme.caltech.edu

*This section is particularly useful for those who would like to implement this method.

I. General procedures

(A) Plasmid construction

All variants described in this study were cloned and expressed using the pET22(b)+ vector (MilliporeSigma, St. Louis, MO). The gene encoding wild-type *Rhodothermus marinus* putative nitric oxide dioxygenase (*Rma* NOD, UniProt ID (1): D0MGT2_RHOM4) was obtained as a single gBlock (Integrated DNA Technologies, Coralville, IA), codon-optimized, and cloned using Gibson assembly (2) into pet22(b)+ with a 6xHisTag appended at the C-terminus. This plasmid was transformed into *E. cloni* ® EXPRESS BL21(DE3) cells (Lucigen, Middleton, WI).

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

ATGGCGCCGACCCTGTCGGAACAGACCCGTCAGTTGGTACGTGCGTCTGTGCCTGCA
CTGCAGAAACACTCAGTCGCTATTAGCGCCACGATGTATCGGCTGCTTTTCGAACGG
TATCCCGAAACGCGGAGCTTATTTGAACTTCCTGAGAGACAGATACACAAGCTTGCG
TCGGCCCTGTTGGCCTACGCCCGTAGTATCGACAACCCATCGGCGTTACAGGCGGCC
ATCCGCCGCATGGTGCTTTCCCACGCACGCGCAGGAGTGCAGGCCGTCCATTATCCG
CTGGTTTGGGAATGTTTGAGAGACGCTATAAAAGAAGTCCTGGGCCCGGATGCCAC
CGAGACCCTTCTGCAGGCGTGGAAGGAAGCCTATGATTTTTTAGCTCATTTACTGTC
TACCAAGGAAGCGCAAGTCTACGCTGTGTTAGCTGAACTCGAGCACCACCACCACC
ACCACTGA

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

MAPTLSEQTRQLVRASVPALQKHSVAISATMYRLLFERYPETRSLFELPERQIHKLASAL
LAYARSIDNPSALQAAIRRMVLSHARAGVQAVHYPLVWECLRDAIKEVLGPDATETLL
QAWKEAYDFLAHLLSTKEAQVYAVLAELEHHHHHH

(B) Protein expression

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

(C) Biocatalytic reaction and assay

In an anaerobic chamber, 10 μL of 400 mM $PhMe_2SiH$ (in acetonitrile) and 10 μL of 400 mM ethyl-2-diazopropanoate (Me-EDA, in acetonitrile) were added to 380 μL whole-cells resuspended in M9-N buffer. The final concentrations in each well were 10 mM Me-EDA and 10 mM $PhMe_2SiH$ in each 400 μL reaction mix. The reaction plate was covered with a foil cover (USA scientific) and shaken at 1000 rpm for 4 hours. 600 μL of cyclohexane was added with a multi-channel pipette to each well to quench the reaction and extract the reaction products, which have been previously characterized (3). Plates were centrifuged to remove cells (3000g, 10 minutes) and enantiomeric excess was measured by running the organic solution on a JACSO 2000 series supercritical fluid chromatography (SFC) system with a Chiralcel OD-H (4.6 mm x 25 cm) chiral column (95% CO_2, 5% isopropanol, 3 minutes). Final variants in main text **Table 1a** and **Table 1b** were expressed and tested in biological triplicate (in addition to the previous protocol of combining 4 replicates). Automatic integration was performed in ChemStation.

(D) Model training

Machine-learning models were trained with sequencing information from MCLAB Inc and enantiomeric data obtained by SFC. To model the data, the following regressors from the superlative scikit-learn package (4) were used: K-nearest neighbors, linear (including Automatic Relevance Detection, Bayesian Ridge, Elastic Net, Lasso LARS, and Ridge), decision trees, random forests (including AdaBoost, Bagging, and Gradient Boosting), and multilayer perceptrons, as it is difficult to know *a priori* which model will best fit the landscape. For example, if the selected positions are truly non-interactive, we can expect much of the landscape's variance to be explained by a linear model. However, for more epistatic landscapes, we must account for this nonlinearity. Therefore, many different model classes were tested, all of which can be run (with hyperparameter optimization) on a personal MacBook Pro in less than one day. The 3 model types with highest Pearson correlation from a Leave-One-Out cross validation (LOO CV) with default hyperparameters were selected for gridsearch hyperparameter optimization. From this gridsearch, the 3 sets of hyperparameters with highest LOO CV Pearson correlation were selected, for a total of 9 models in order to capture different characteristics of the landscape with relatively low accuracy models. The models were retrained on the full dataset and used for predicting a restricted library, discussed in the section below.

(E) Model predictions

Directly synthesizing the DNA encoding the top variants is quite expensive. Therefore, we interpret our models' predictions by the frequency of each amino acid's occurrence in a top fraction (the top 1000) of the library, which we are able to encode efficiently with degenerate codon libraries. An example is shown in **supplementary Table 3**. For cloning purposes, at this point the sequence information predicted from the models is lost, as each position is considered independently to reduce DNA synthesis and subcloning costs. Additionally, we elected to include all 20 amino acids in the predictions even though less than 20 were encoded in the input libraries, to provide an estimate for when the models may be predicting high fitness based on

mutations at other positions. A full description of this step is provided with the accompanying **supplementary Table 3**.

(F) Experimental validation of predictions

The top amino acids at each position are encoded by degenerate codons identified by SwiftLib (5). All 9 models are considered when choosing amino acids to encode, in case some models are capturing different characteristics of the sequence-function relationship. While the optimal combinations of amino acids identified by the model are retained in this library, there may be non-optimal combinations that result from this procedure. However, we have developed this method to balance these experimental costs with being able to access the restricted libraries. The degenerate codons used to encode the predicted libraries are shown in **supplementary Figure 3.** The predicted libraries were tested in the same manner as above.

Table 1: Starting activity

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

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

Table 2: Modeling statistics

The accuracy for the 9 models of each Set, as well as the average values of the 9 models, is shown for the data obtained by screening the predicted libraries. The corresponding predicted versus measured values can be found in **supplementary Figure 3**.

Table 2a: Test errors for Set I from predicted (*R*)- and (*S*)- libraries

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

Table 2b: Test errors for Set II from predicted (*R*)- library

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

Table 2c: Test errors for Set II from predicted (*S*)- library

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

Table 3: Sample prediction frequency table – position Set II (*S*)-
A sample predicted frequency table from position Set II for the (*S*)-enantiomer is provided below. At this step, the exact combinations predicted by the machine learning models are lost, and instead interpreted as frequencies at individual amino acid positions to encode with degenerate codons. The alternative (ordering and cloning the top sequences individually) can be quite expensive, but tractable if screening costs significantly outweigh DNA synthesis costs. All 20 canonical amino acids are enumerated, although each library does not contain all 20 in the input library. Amino acids that are not present in the input library, but predicted to be high-functioning, can be used as indicators for when the frequencies may be relying on amino acids at other positions to make predictions. In other words, they serve as cut-offs above which the amino acids should be considered.

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

For example, the first amino acids that were not present in the input dataset for each of the three positions are Q, Q, M as NDT encodes: {N, S, I, H, R, L, D, G, V, Y, C, F}. The amino acids occurring significantly more frequently in the top 1000 sequences are then [Y, N, R, G], [V, F, I], and [I, V, L, S, F]. This process is repeated for the top 3 models determined by default hyperparameters, and then 3 hyperparameter sets are used for each optimal, for a total of 9 models (in an attempt to capture different portions of the landscape with inaccurate models). In the described approach, this step can be tuned to consider more or less sequences (such as the top 20% or top 1%) depending on the protein engineer's discretion considering the following: screening throughput, sequencing cost, cloning capabilities, desired fitness improvement, model accuracy, theoretical size of the predicted library, ease of encoding with codon degeneracy, and an interpretation of the landscape (how many variants are expected to be near the fitness peak). As DNA synthesis costs continue to fall, the ideal is to be able to sample the top sequences

directly (as was simulated in our study with data from a full recombination library of 4 positions) without resorting to this approach to interpret the models' predictions with degenerate codons.

Supplementary Table 4: Relative activity compared to starting sequence

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

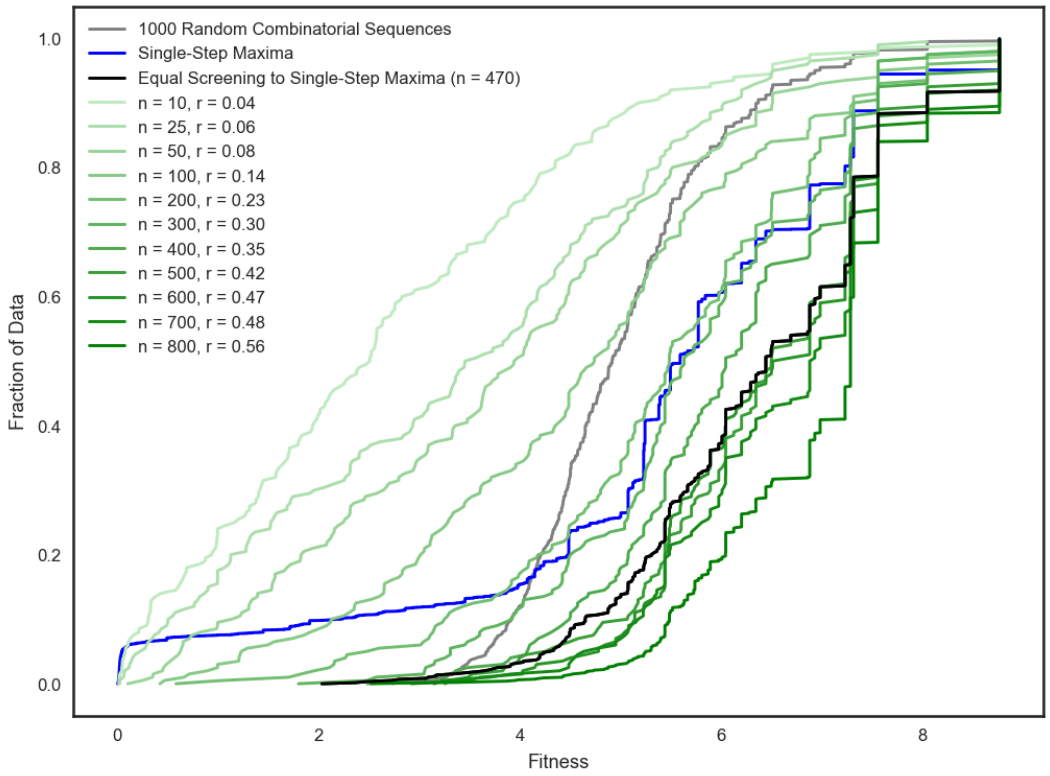
Set I: Residues 32, 46, 56, and 97

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

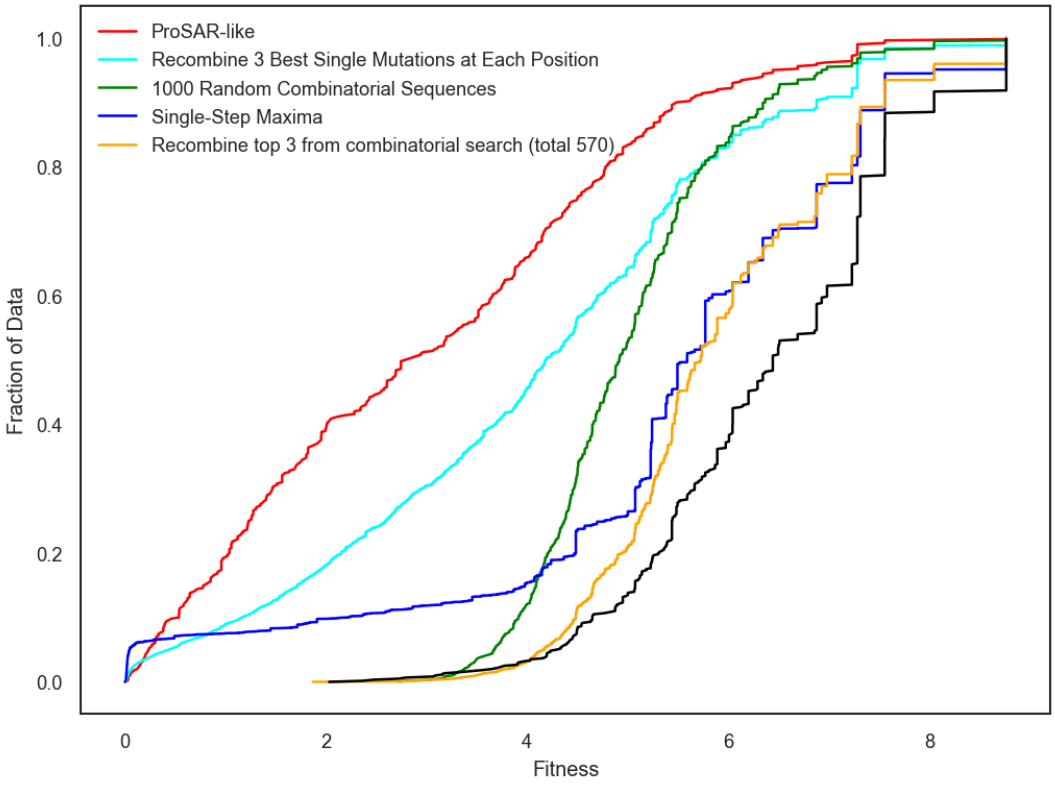
Set II: Residues 49, 51, and 53

	Input Variants						Predicted Variants					
	Residue			**Enantioselectivity**		**Activity compared to KFLL**	**Residue**			**Enantioselectivity**		**Activity compared to KFLL**
	49	51	53	% *S*-isomer	% *R*-isomer		49	51	53	% *S*-isomer	% *R*-isomer	
(*S*)-selective From VCHV	P	R	I	**93**	7	2.5	Y	V	V	**97**	3	2.8-fold
	Y	V	F	**93**	7	1.5	P	V	I	**96**	4	3.2-fold
	N	D	V	**87**	13	0.8	P	V	V	**96**	4	3.1-fold
(*R*)-selective From GSSG	P	R	I	19	**81**	2.7	P	R	L	11	**89**	2.2-fold
	N	S	Y	22	**78**	0.8	P	G	L	13	**87**	2.1-fold
	N	I	I	22	**78**	1.0	P	F	F	15	**85**	2.2-fold

Supplementary Figure 1A: Highest fitnesses found with less accurate models



Supplementary Figure 1B: Highest fitnesses found with other DE approaches



Empirical Cumulative Distribution Functions (eCDFs) are shown for increasing amount of data input in **Supp. Fig. 1A**, and for the various evolutionary methods established in the main text in **Supp. Fig. 1B**. In **Supp. Fig. 1A**, two hundred simulated evolutions are tested for each of the machine learning-assisted methods, with the exception of N=470, where 600 are tested. Lines shifted toward the right are more likely to identify sequences with higher fitness. The cumulative fraction is shown on the ordinate axis, and fitness value on the abscissa. The highest fitness value from the top 100 sequences (roughly the smallest batch size, as screening is typically done in 96 well plates) from each model trained with N sequences is shown to demonstrate the effect of increased training data. Therefore, the total screening burden for each line is N + 100. With 570 sequences measured (in black), the machine learning-assisted evolution approach reaches the global optimum fitness value in 8.4% of simulations, compared to 4.9% of all starting sequences (in blue). The machine learning-assisted evolution approach only requires between 300 and 400 total tested sequences to perform similarly to directed evolution (570 sequences). Therefore, the directed evolution approach requires about 42% more variants tested to achieve similar results on this landscape. However, perhaps a more important metric is the expected fitness value obtained by each method, summarized below.

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

Other controls are included in **Supp. Fig. 1B for** random combinatorial sequences, from which the highest fitness from 1000 random samples is provided (in gray), two different methods of recombination (in cyan and gold), and a ProSAR-like algorithm (in red). In cyan, recombination from the top 3 single mutants at each position from a reference parent are shown. The top 3 mutants from a random combinatorial search of all positions is shown in gold (with an average of 570 sequences searched).

Our implementation of ProSAR is based on the Partial Least Squares (PLS) algorithm for a linear model for point mutations established by Fox and coworkers (6, 7). Specifically, the PLS implementation by scikit-learn is trained with data from 569 random sequences (optimized over the number of components kept). From the PLS decomposition, the coefficients for the linear contribution from each mutation is determined, and the most positive mutation at each position is

kept. We call this approach “ProSAR-like”, as the exact implementation of ProSAR can be fairly subjective (see Supporting Information (Detailed description of a round): *Improving catalytic function by ProSAR-driven enzyme evolution* by Fox *et al.* (7)).

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

Supplementary Figure 2: **Predicted vs measured values for all libraries**
The predicted versus measured values for sequence-verified variants in the predicted libraries are shown for each library. Figure 2A contains predicted values for position Set I. Figure 2B contains predicted values for position Set II from GSSG, and Figure 2C for Set II from VCHV. A linear regression is shown for these values.

Figure 2A: Predicted vs measured values for *ee* from Set I

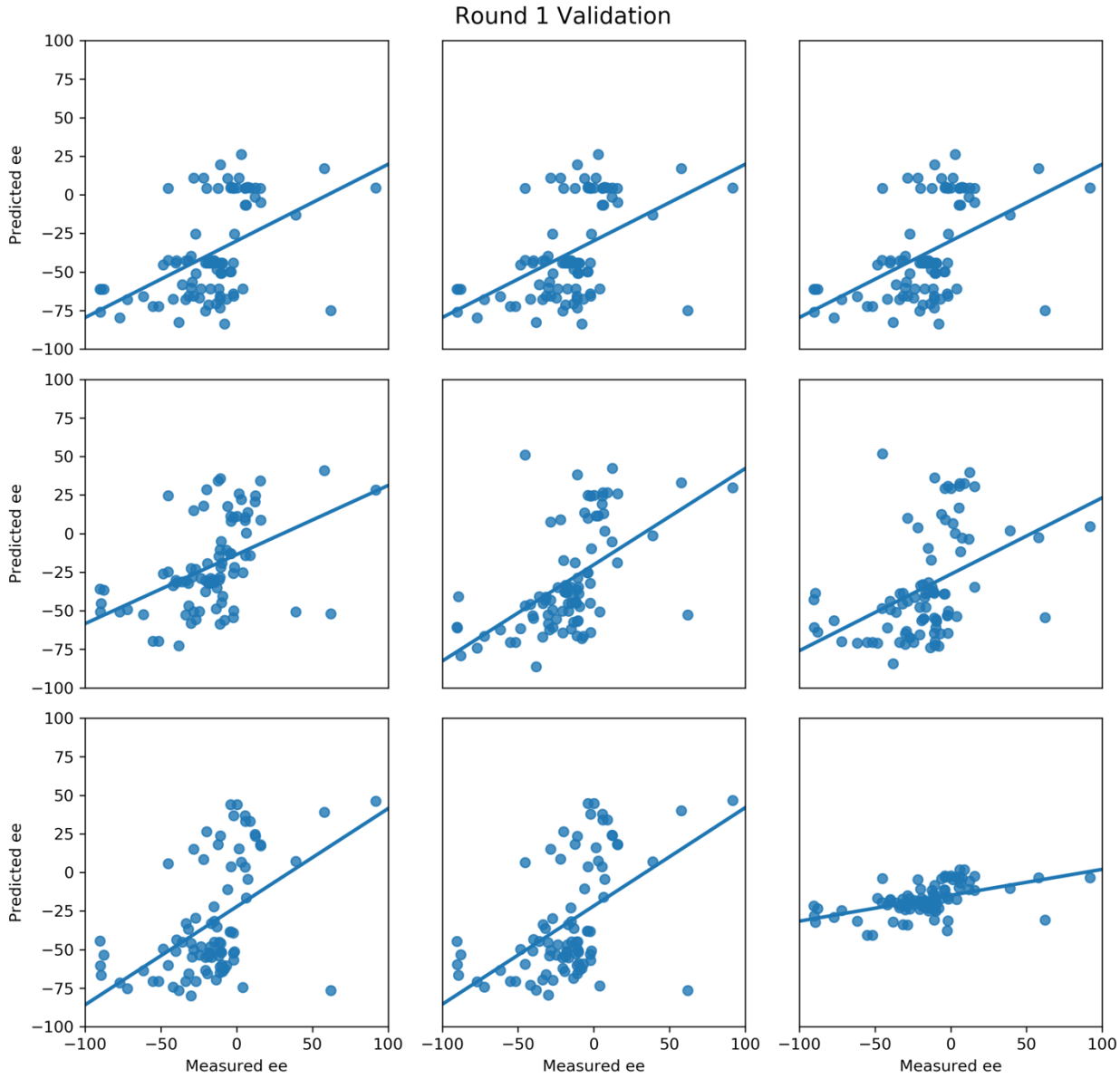


Figure 2B: Predicted vs measured values for *ee* from position Set II from GSSG

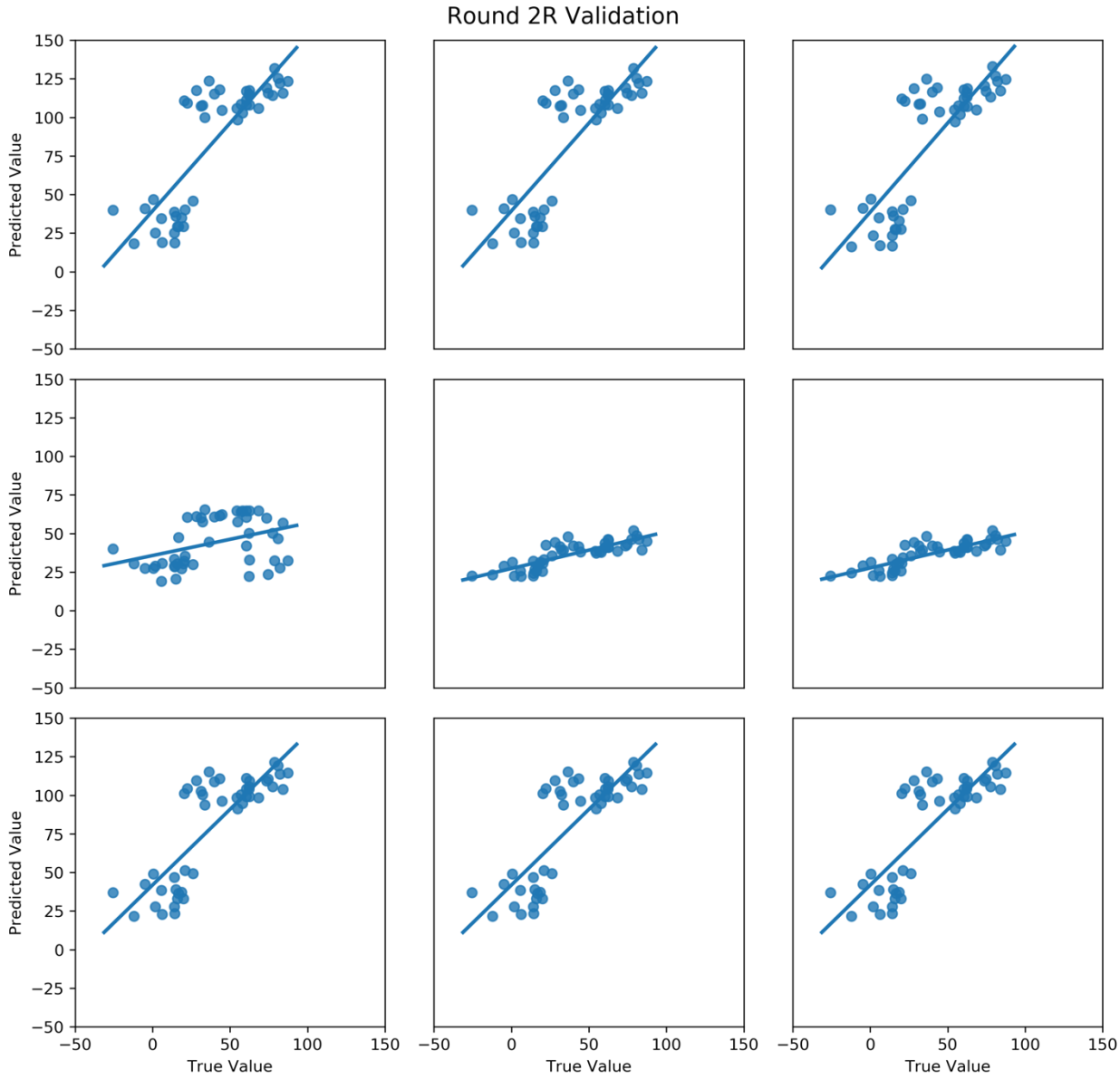
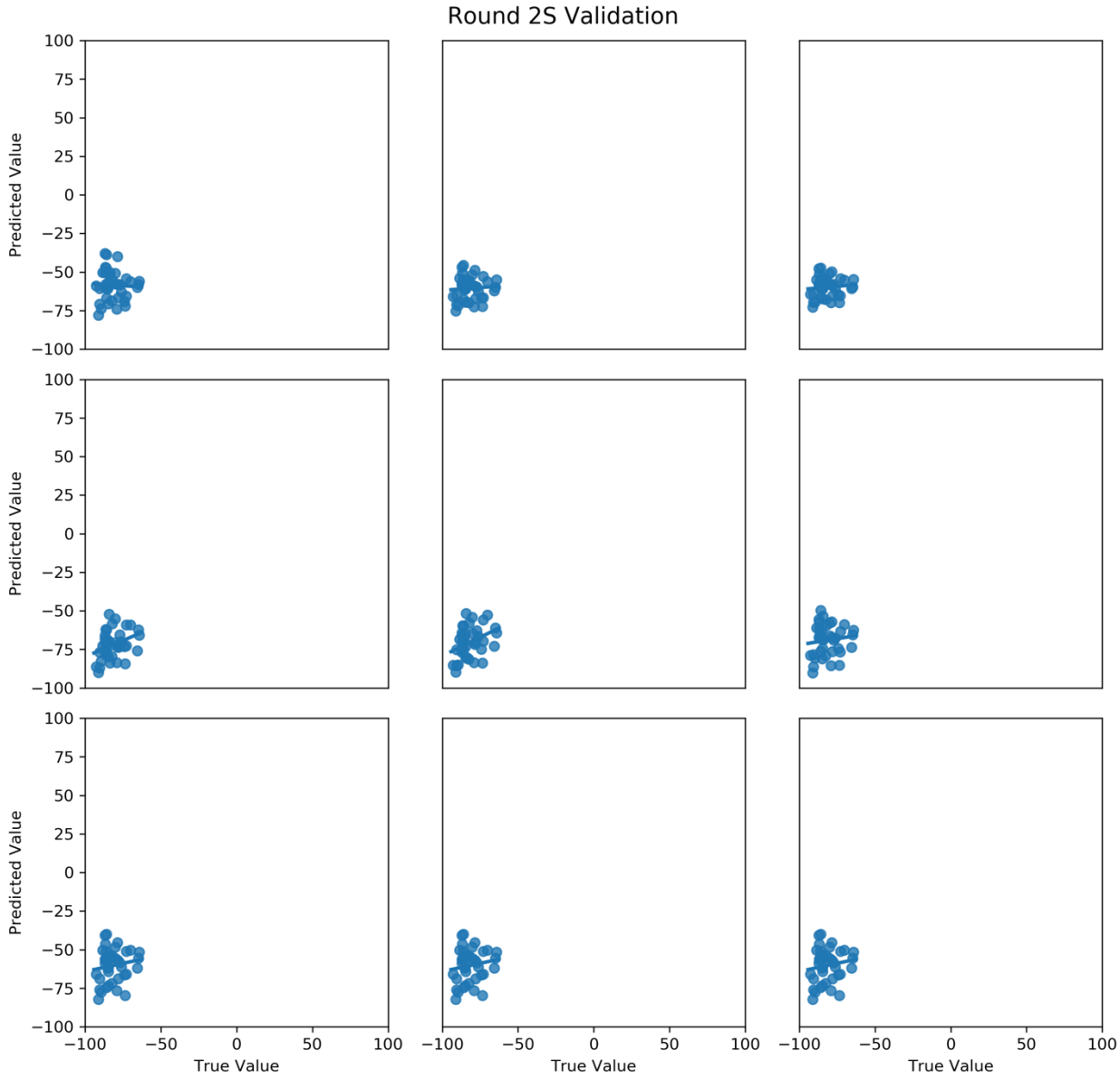


Figure 2C: Predicted vs Measured Values for *ee* from Position Set II from VCHV

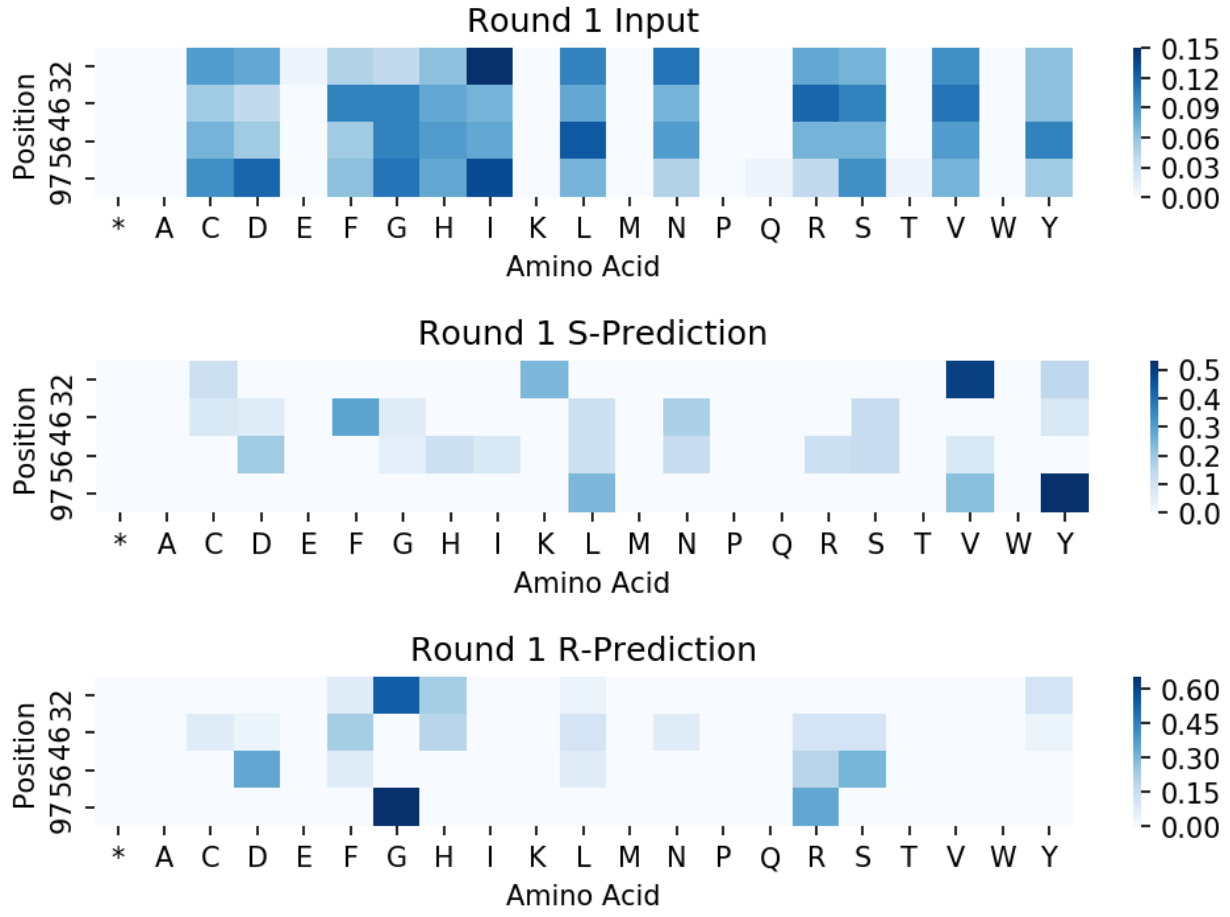


IV. Input sequences versus encoded predictions

A bias that is not present in the empirical landscape study is in the ratio of sequences that are transformed into the host organism. This ratio can be significantly altered due to cloning biases. Therefore, heat maps of encoded amino acids are shown for each round comparing the input and predicted libraries. (These ratios are often represented with sequence logo maps, which are better visualizations when a few amino acids dominate.)

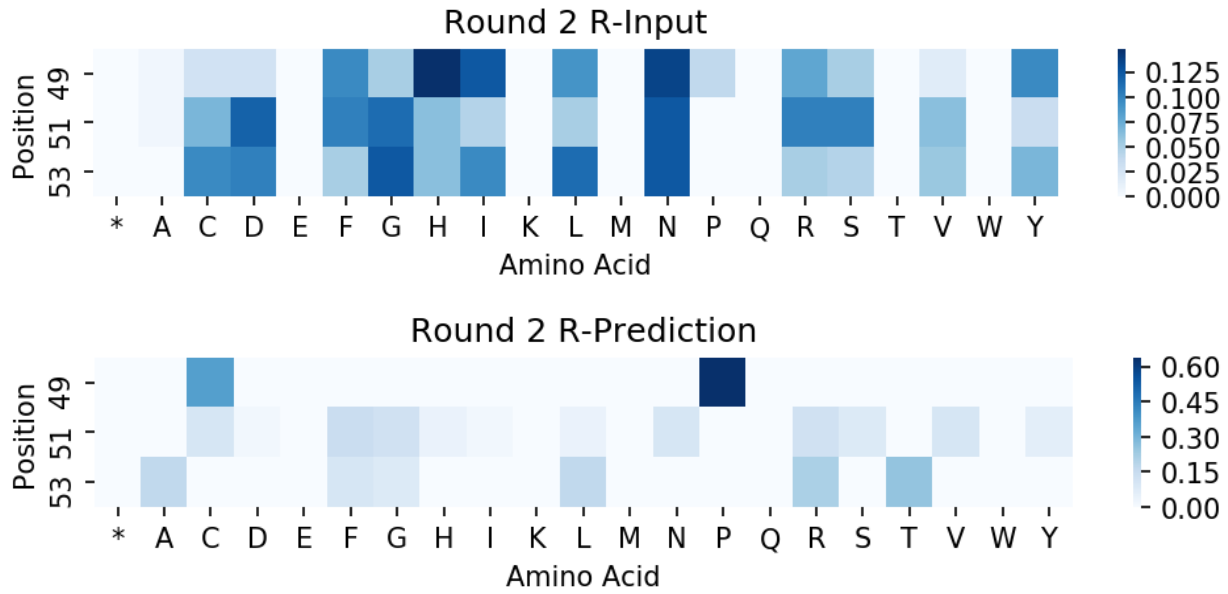
The input libraries are NDT libraries, which represent N, S, I, H, R, L, D, G, V, Y, C, F. Input libraries also contain proline at position 49 from WT. The degenerate codons used to encode amino acids at each position are provided for reference. Sequence-function data is available on Protabank (8). In these tables, residues in bold were not part of the predicted library but had to be included with the degenerate codon cloning method.

Figure 3A: Input versus predicted sequences for modeling position Set I.



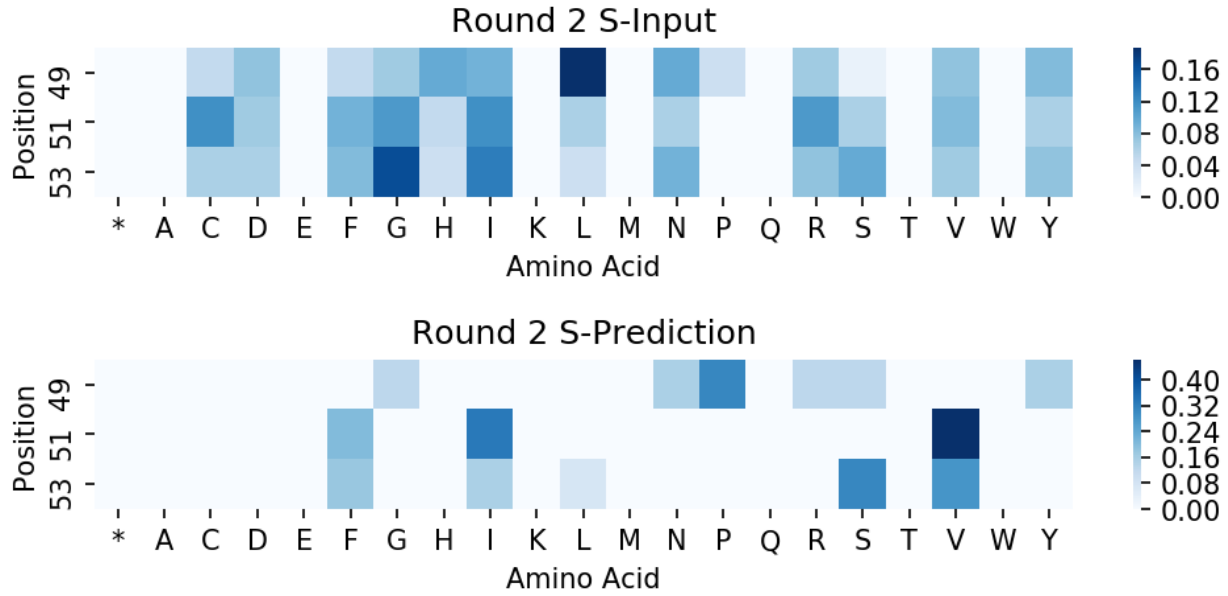
1R Predictions			1S Predictions		
Position	Codon	Encoded	Position	Codon	Encoded
32	AAA; GTA; TRC	K; V; C, Y	32	GGA; YWC	G; F, H, L, **Y**
46	DRC; TTM	C, D, G, N, S, Y; F, L	46	HDC	C, F, H, I, L, N, **R**, S, Y
56	VDC	**D**, G, H, I, L, N, R, S, V	56	GAC; YBC	D; C, F, L, **P**, R, S
97	STA; TAC	L, V;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

Figure 3B: Input versus predicted sequences for modeling Set II from GSSG.



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

Figure 3C: Input versus predicted sequences for modeling Set II from VCHV.



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

V. Library coverage

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

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

$$P_i = 1 - (1 - f_i)^N$$

Rearranged to give

$$N = \frac{\ln (1 - P_i)}{\ln (1 - f_i)}$$

As stated previously, a typical desired library coverage is $P_i = 0.95$ for 95% library coverage, but the choice of codons can have a strong effect on the value of f_i. Assuming equal representation of the 19 codons gives $N \approx 55.4$, or 554 variants for the 10 libraries needed. However, the authors of the landscape used NNS/NNK codons, which encode for 20 amino acids with 32 codons. The least frequent amino acid encoded with these codons (methionine) occurs at a frequency of 1/32, requiring $N \approx 94.4$, or 944 variants. A typical balance between balancing the degenerate codon complexity and amino acid coverage that protein engineers employ is the use of NDT/VHG/TGG codons, also known as the 22c-trick (10), in which methionine occurs 1/22 times for $N \approx 64.4$, or 644 variants over 10 libraries.

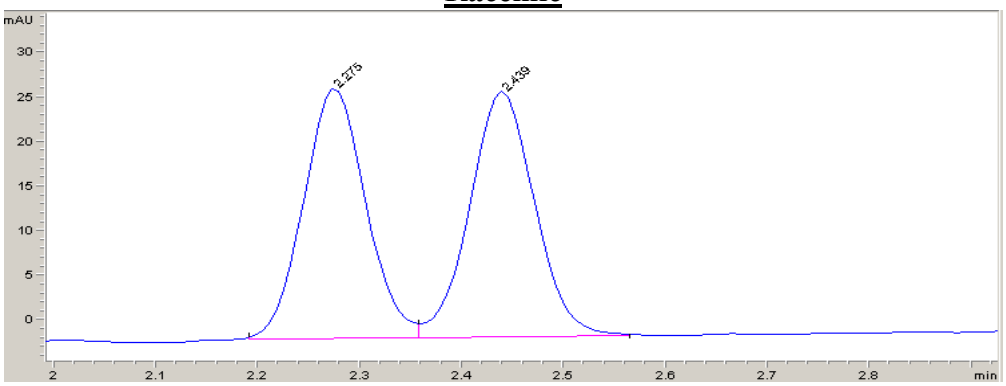
From a protein engineer's perspective, a comparison to 644 variants is likely the most pertinent in **Supp Figure 1**. However, to provide DE alone with a stronger baseline, we have used 570 variants, obtained from applying the 3-fold oversampling rule (9) to 10 libraries containing 19 desired variants each. as a comparison in the main text. In any case, we empirically observe that the single-mutation walk performs similarly to the ML approach at 300-400 variants (**Supp Figure 1)**, which is significantly less than any of the numbers presented here.

VI. Chiral SFC traces for racemic and enzymatically synthesized organosilicon products

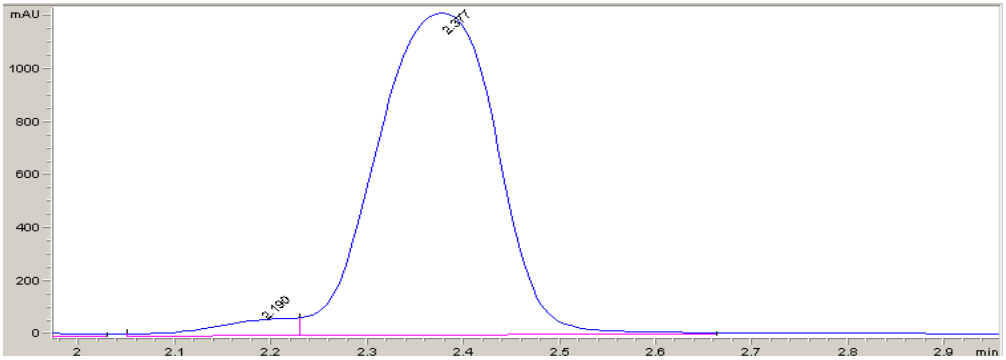
All the *ee* values of synthesized organosilicon products were determined using automatic peak integration from chiral SFC. The traces for racemic and enzymatic products are shown below. The absolute configuration of products was previously determined (3).

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

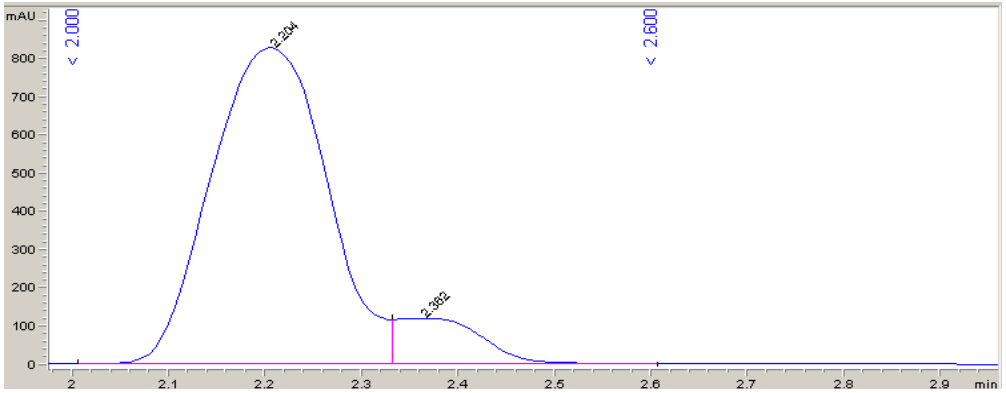
Racemic



Variant VCHVYVV



Variant GSSGPRL



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

VII. Experimental uncertainty in best *Rma* NOD variants

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

Activity is significantly improved over starting variant KFLL

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

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

Enantioselectivity in Set II is significantly improved over starting variant VCHV

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

Enantioselectivity in Set II is significantly improved over starting variant GSSG

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

VIII. Model performance and selection

The LOO Pearson r of the regressors with default hyperparameters and the models ultimately selected are shown for each round.

Set I Models

0.512212499	GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
0.478097911	RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)
0.460760125	LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000, random_state=None, tol=0.0001, verbose=0)
0.447166856	ARDRegression(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.001, verbose=False)
0.423793421	KernelRidge(alpha=1, coef0=1, degree=3, gamma=None, kernel='linear', kernel_params=None)
0.419172462	BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, tol=0.001, verbose=False)
0.406655665	BaggingRegressor(base_estimator=None, bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)
0.396791771	LassoLarsCV(copy_X=True, cv=None, eps=2.2204460492503131e-16, fit_intercept=True, max_iter=500, max_n_alphas=1000, n_jobs=1, normalize=True, positive=False, precompute='auto', verbose=False)
0.37899373	DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')
0.371734032	SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01, fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling', loss='squared_loss', n_iter=5, penalty='l2', power_t=0.25, random_state=None, shuffle=True, verbose=0, warm_start=False)
0.366256085	KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')
0.338423931	ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5, max_iter=1000, normalize=False, positive=False, precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
0.202908183	AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear', n_estimators=50, random_state=None)
-0.082371549	LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
-0.766587492	NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False)

Set I Models Selected

- ARDRegression(alpha_1=0.01, alpha_2=0.1, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.01, lambda_2=0.01, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- ARDRegression(alpha_1=0.01, alpha_2=0.01, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.01, lambda_2=0.01, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- ARDRegression(alpha_1=0.01, alpha_2=0.0001, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.01, lambda_2=0.01, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- GradientBoostingRegressor(alpha=0.3, criterion='mse', init=None,learning_rate=0.9, loss='quantile', max_depth=3,max_features=None, max_leaf_nodes=None,min_impurity_split=1e-07, min_samples_leaf=1,min_samples_split=2, min_weight_fraction_leaf=0.0,n_estimators=500, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
- GradientBoostingRegressor(alpha=0.5, criterion='mse', init=None,learning_rate=0.7, loss='huber', max_depth=10,max_features=None, max_leaf_nodes=None,min_impurity_split=1e-07, min_samples_leaf=1,min_samples_split=2, min_weight_fraction_leaf=0.0,n_estimators=1000, presort='auto', random_state=None,subsample=1.0, verbose=0, warm_start=False)
- GradientBoostingRegressor(alpha=0.5, criterion='mse', init=None,learning_rate=0.7, loss='huber', max_depth=10,max_features=None, max_leaf_nodes=None,min_impurity_split=1e-07, min_samples_leaf=1,min_samples_split=2, min_weight_fraction_leaf=0.0,n_estimators=100, presort='auto', random_state=None,subsample=1.0, verbose=0, warm_start=False)
- LinearSVR(C=50, dual=True, epsilon=0, fit_intercept=True,intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000,random_state=None, tol=0.0001, verbose=0)
- LinearSVR(C=50, dual=True, epsilon=0.1, fit_intercept=True,intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000,random_state=None, tol=0.0001, verbose=0)
- LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True,intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000,random_state=None, tol=0.0001, verbose=0)

Set 2(*S*) Models

0.609011	ARDRegression(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.001, verbose=False)
0.60823	NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False)
0.598652	LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000, random_state=None, tol=0.0001, verbose=0)
0.587369	KernelRidge(alpha=1, coef0=1, degree=3, gamma=None, kernel='linear', kernel_params=None)
0.584004	BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, tol=0.001, verbose=False)
0.578776	GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
0.577483	LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
0.564136	BaggingRegressor(base_estimator=None, bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)
0.549607	RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)
0.503091	MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)
0.499812	DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')
0.438121	LassoLarsCV(copy_X=True, cv=None, eps=2.2204460492503131e-16, fit_intercept=True, max_iter=500, max_n_alphas=1000, n_jobs=1, normalize=True, positive=False, precompute='auto', verbose=False)
0.437293	SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01, fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling', loss='squared_loss', max_iter=None, n_iter=None, penalty='l2', power_t=0.25, random_state=None, shuffle=True, tol=None, verbose=0, warm_start=False)
0.430372	KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')

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

Set 2(*S*) Models Selected

- GradientBoostingRegressor(alpha=0.1, criterion='mse', init=None, learning_rate=0.7, loss='lad', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
- GradientBoostingRegressor(alpha=0.5, criterion='friedman_mse', init=None, learning_rate=0.9, loss='quantile', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
- GradientBoostingRegressor(alpha=0.3, criterion='friedman_mse', init=None, learning_rate=0.3, loss='quantile', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
- ARDRegression(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.1, lambda_2=1e-06, n_iter=3000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- ARDRegression(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.1, lambda_2=1e-06, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- ARDRegression(alpha_1=0.1, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.1, lambda_2=1e-06, n_iter=3000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- LinearSVR(C=50, dual=True, epsilon=0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000, random_state=None, tol=0.0001, verbose=0)
- LinearSVR(C=100, dual=True, epsilon=0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000, random_state=None, tol=0.0001, verbose=0)
- LinearSVR(C=1000, dual=True, epsilon=0, fit_intercept=True, intercept_scaling=1.0, loss='squared_epsilon_insensitive', max_iter=10000, random_state=None, tol=0.0001, verbose=0)

Set 2(*R*) Models

0.651306	GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
0.635081	ARDRegression(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, threshold_lambda=10000.0, tol=0.001, verbose=False)
0.631197	BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300, normalize=False, tol=0.001, verbose=False)
0.626982	KernelRidge(alpha=1, coef0=1, degree=3, gamma=None, kernel='linear', kernel_params=None)
0.625465	AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear', n_estimators=50, random_state=None)
0.61362	NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='auto', kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False)
0.612285	LassoLarsCV(copy_X=True, cv=None, eps=2.2204460492503131e-16, fit_intercept=True, max_iter=500, max_n_alphas=1000, n_jobs=1, normalize=True, positive=False, precompute='auto', verbose=False)
0.608155	RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)
0.595166	MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)
0.583177	BaggingRegressor(base_estimator=None, bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)
0.552808	LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000, random_state=None, tol=0.0001, verbose=0)
0.542432	SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01, fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling', loss='squared_loss', max_iter=None, n_iter=None, penalty='l2', power_t=0.25, random_state=None, shuffle=True, tol=None, verbose=0, warm_start=False)
0.479498	DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

0.473718	KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')
0.242713	LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

Set 2(*R*) Models Selected

- GradientBoostingRegressor(alpha=0.1, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
- GradientBoostingRegressor(alpha=0.5, criterion='mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=500, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
- GradientBoostingRegressor(alpha=0.7, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', random_state=None, subsample=1.0, verbose=0, warm_start=False)
- ARDRegression(alpha_1=1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=10000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- ARDRegression(alpha_1=1, alpha_2=0.0001, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=10000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- ARDRegression(alpha_1=1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=0.0001, lambda_2=0.1, n_iter=10000, normalize=False, threshold_lambda=10000.0, tol=0.0001, verbose=False)
- BayesianRidge(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=10000, normalize=False, tol=0.0001, verbose=False)
- BayesianRidge(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=3000, normalize=False, tol=0.0001, verbose=False)
- BayesianRidge(alpha_1=0.1, alpha_2=1e-08, compute_score=False, copy_X=True, fit_intercept=True, lambda_1=1e-08, lambda_2=0.1, n_iter=300, normalize=False, tol=0.0001, verbose=False)

IX. Supplemental references

1. Bateman A, et al. (2017) UniProt: The universal protein knowledgebase. *Nucleic Acids Res*. doi:10.1093/nar/gkw1099.
2. Gibson DG, et al. (2009) Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nat Methods*. doi:10.1038/nmeth.1318.
3. Kan SBJ, Lewis RD, Chen K, Arnold FH (2016) Directed evolution of cytochrome c for carbon–silicon bond formation: Bringing silicon to life. *Science (80-)* 354(6315):1048–1051.
4. Pedregosa F, Varoquaux G (2011) Scikit-learn: Machine learning in Python. *J Mach Learn Res* 12:2825–2830.
5. Jacobs TM, Yumerefendi H, Kuhlman B, Leaver-Fay A (2015) SwiftLib: rapid degenerate-codon-library optimization through dynamic programming. *Nucleic Acids Res* 43(5):e34.
6. Fox RJ, et al. (2003) Optimizing the search algorithm for protein engineering by directed evolution. *Protein Eng* 16(8):589–597.
7. Fox RJ, et al. (2007) Improving catalytic function by ProSAR-driven enzyme evolution. *Nat Biotechnol* 25(3):338–44.
8. Wang CY, et al. (2018) ProtaBank : A repository for protein design and engineering data. 1–35.
9. Reetz MT, Kahakeaw D, Lohmer R (2008) Addressing the numbers problem in directed evolution. *ChemBioChem* 9(11):1797–1804.
10. Kille S, et al. (2013) Reducing codon redundancy and screening effort of combinatorial protein libraries created by saturation mutagenesis. *ACS Synth Biol* 2(2):83–92.
11. Bosley AD, Ostermeier M (2005) Mathematical expressions useful in the construction , description and evaluation of protein libraries. *Biomol Eng* 22:57–61.