THE UNIVERSITY OF CHICAGO

DATA-DRIVEN INTERPRETATION AND DESIGN OF ORTHOLOGS AND PARALOGS OF A SIGNALING PROTEIN

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF CHEMISTRY

BY
XINRAN LIAN

CHICAGO, ILLINOIS
MARCH 2024

I dedicate this thesis to my two esteemed supervisors, Rama Ranganathan and Andrew Ferguson, whose mentorship and guidance have been invaluable throughout my PhD journey. Your encouragement, knowledge, and insights have been instrumental in shaping my research and pushing me to achieve more than I thought possible. I am truly grateful for the opportunities you have provided me and for always challenging me to think critically and creatively.

I also dedicate this thesis to my collaborators, who have contributed to my research in many ways. Your expertise and support have been invaluable, and I am fortunate to have had the opportunity to work with such talented and dedicated individuals. Together, we have tackled complex problems and achieved meaningful results that will have a lasting impact on the scientific community.

"Science knows no country, because knowledge belongs to humanity, and is the torch which illuminates the world." - Louis Pasteur

TABLE OF CONTENTS

LIST OF FIGURES

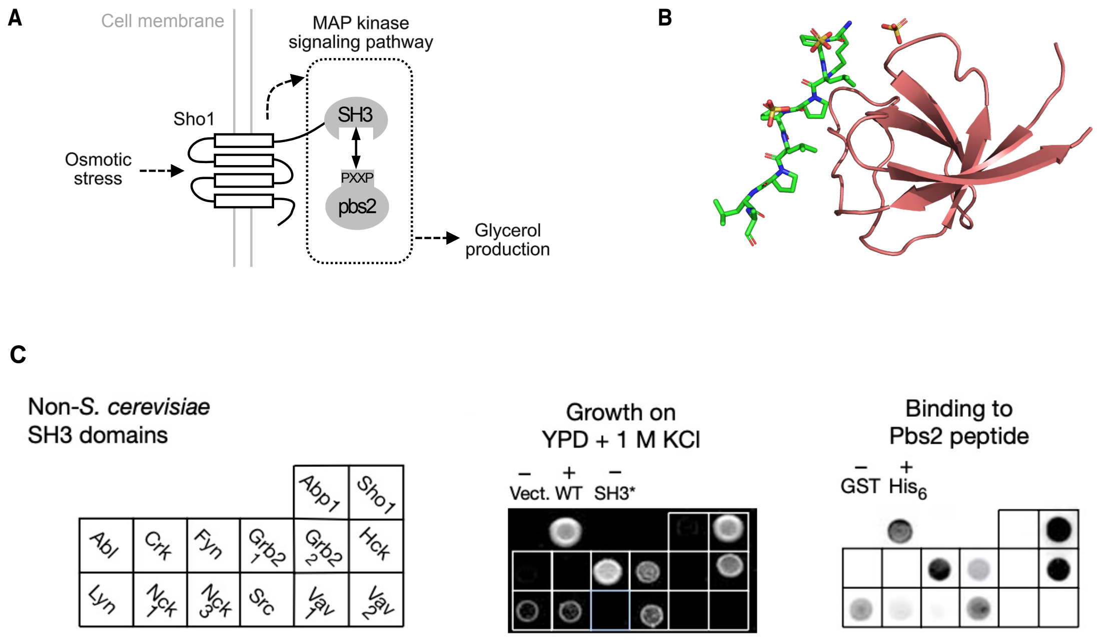


Figure 1: **Positive and negative design of the osmosensing function of Sho1^{SH3}.** (**A**) Binding between the Sho1^{SH3} domain and its target sequence in the Pbs2 MAP kinase kinase mediates responses to fluctuations in external osmotic pressure by controlling the production of internal osmolytes.(**B**)A structure of the *S. cerevisiae* Sho1^{SH3} domain (PDB 2VKN) in complex with the Pbs2 peptide ligand (yellow stick bonds). SH3 domains are protein interaction modules that bind to polyproline containing target ligands. (**C**) The growth of cells that contain Sho1 chimaeras with swapped SH3 domains on high-osmolarity medium, along with the SH3 binding arrays presented in Zarrinpar et al's paper[Zarrinpar et al., 2003]. The left side of the figure displays the arrangement of the chimaeras, and the subscript denotes the domain number in multidomain proteins.

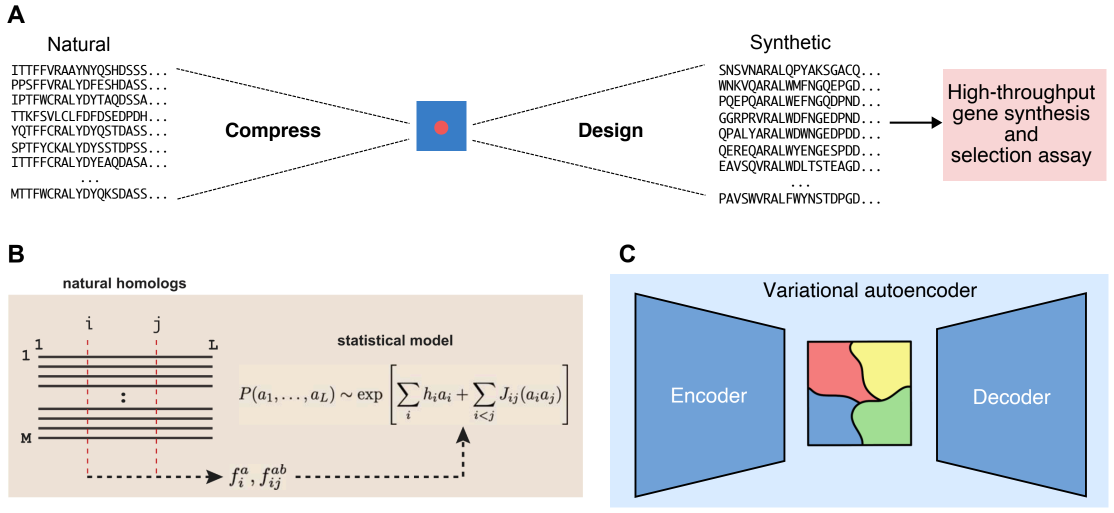


Figure 2: **Principle of the data-driven models for protein design**
(**A**) Schematic of evolutionary-based data-driven generative models, consisting of a compression step that maps a sequence alignment of natural homologs to a low-dimensional parameter space (blue box), and a de-compression step which design protein sequences from the parameters. The protein sequences are then synthesized and tested by high-throughput gene synthesis and selection assays.(**B**) Principle of the bmDCA model [Russ et al., 2020]. MSA of M natural homologs provides empirical first- and second-order statistics of amino acids ($f_i^a; f_{ij}^{ab}$), which are used to infer a statistical model with the bmDCA method. The probability of sequence $a = (a_1; \ldots; a_L)$ is an exponential function of a Hamiltonian, or statistical energy, parameterized by intrinsic fields $h_i(a)$ and couplings $J_{ij}(a, b)$ acting on amino acids. (**C**) Structure of the VAE models. The encoder encodes protein sequences into a continuous latent representation and the decoder subsequently decodes them back to the original space, thereby learning to generate novel protein sequences with similar properties to those in the training dataset.

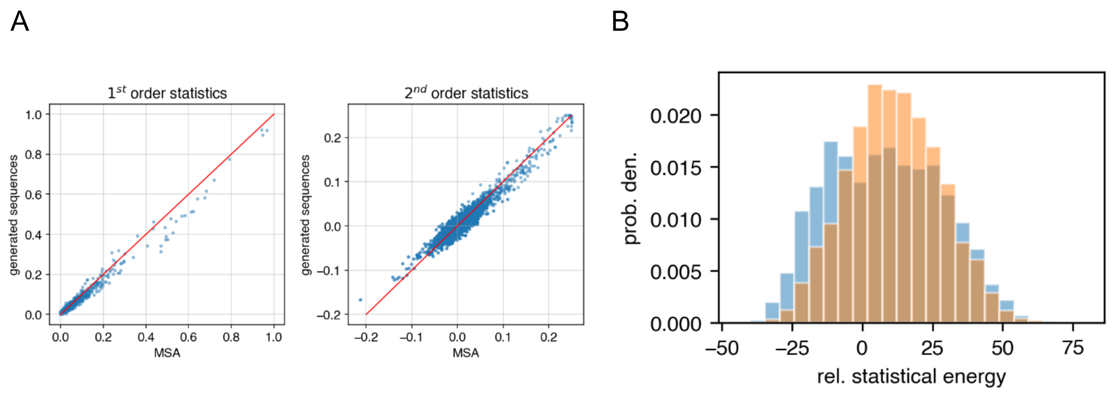


Figure 3: **Analysis of bmDCA sampled sequences**
(**A**) Parity plot of the one- and two-body amino acid frequencies computed over the 5299 natural sequences and an ensemble of 3740 bmDCA designed synthetic variants sampled at $T = 1$. The red lines indicate the identity relationship. The excellent agreement (one-body: ρ_{Pearson}=0.99, n= 5000, $p < 1 \times 10^{-307}$; two-body: ρ_{Pearson}=0.82, n= 5000, $p < 1 \times 10^{-307}$) demonstrates the validity of the bmDCA model in accurately learning and reproducing the one- and two-body amino acid frequencies. (**B**) Distribution of statistical energies for natural (blue) and bmDCA designed sequences sampled at $T = 0.9$ (orange). The two distributions lie on the same range and with a similar distribution.

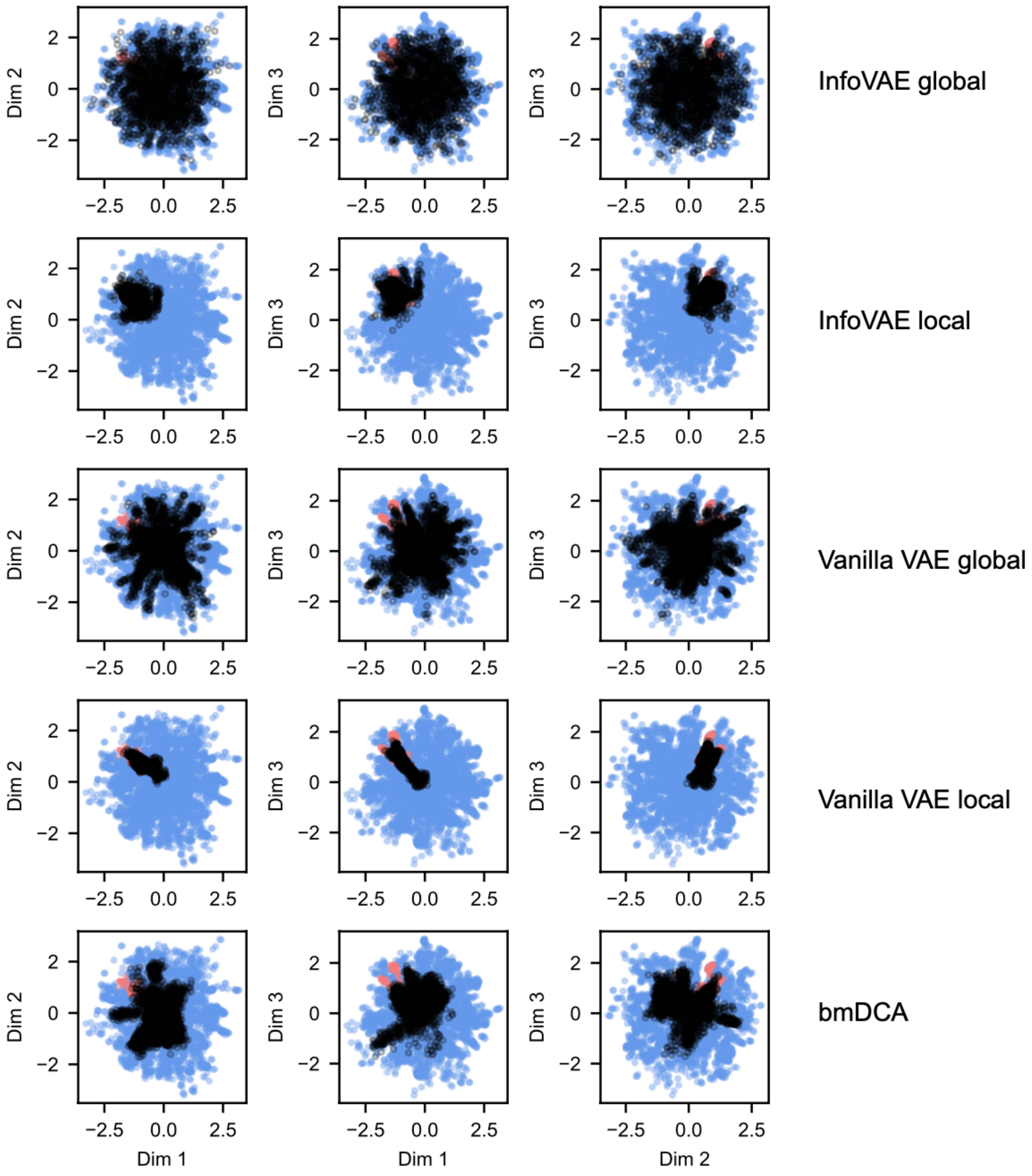


Figure 4: **Projection into the 3D InfoVAE latent space of the synthetic sequences** Projection into the 3D InfoVAE latent space of the synthetic sequences designed by global (n=2000) and local (n=987) sampling over the InfoVAE latent space, global (n=3984) and local (n=896) sampling over the vanilla VAE latent space, and MCMC sampling from the bmDCA generative model (n=3740). Designed sequences are shown in black and superposed on the n=170 natural SH3 homologs that possess high-r.e. scores and rescue osmosensing function (red) and the remaining n=5129 that fail to rescue (blue).

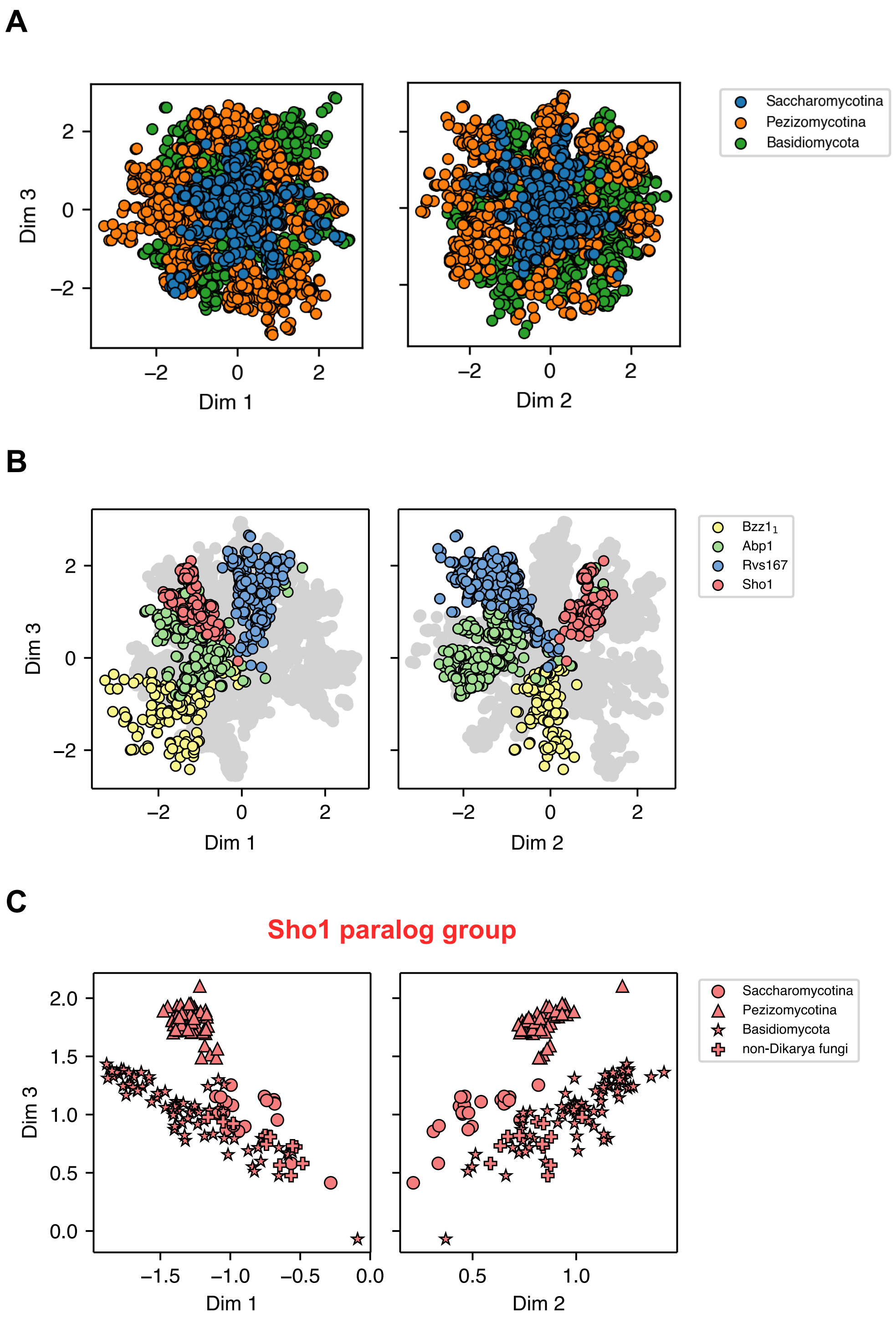


Figure 5: **The InfoVAE latents learns a nested hierarchical partitioning of natural fungal SH3 homologs by function and phylogeny.**
(A) InfoVAE 3D latent space embedding of the 5299 natural SH3 homologs annotated by the three main fungal phylogeny groups. (B) Annotation by paralog group and phylogenetic annotation within the Sho1 paralog cluster (red): Saccaromycotina (circle), Pezizomycotina (triangle), Basidiomycota (star) and non-dikarya (plus). Analogous plots for the remaining 3 paralog groups are presented in Figs. 6.

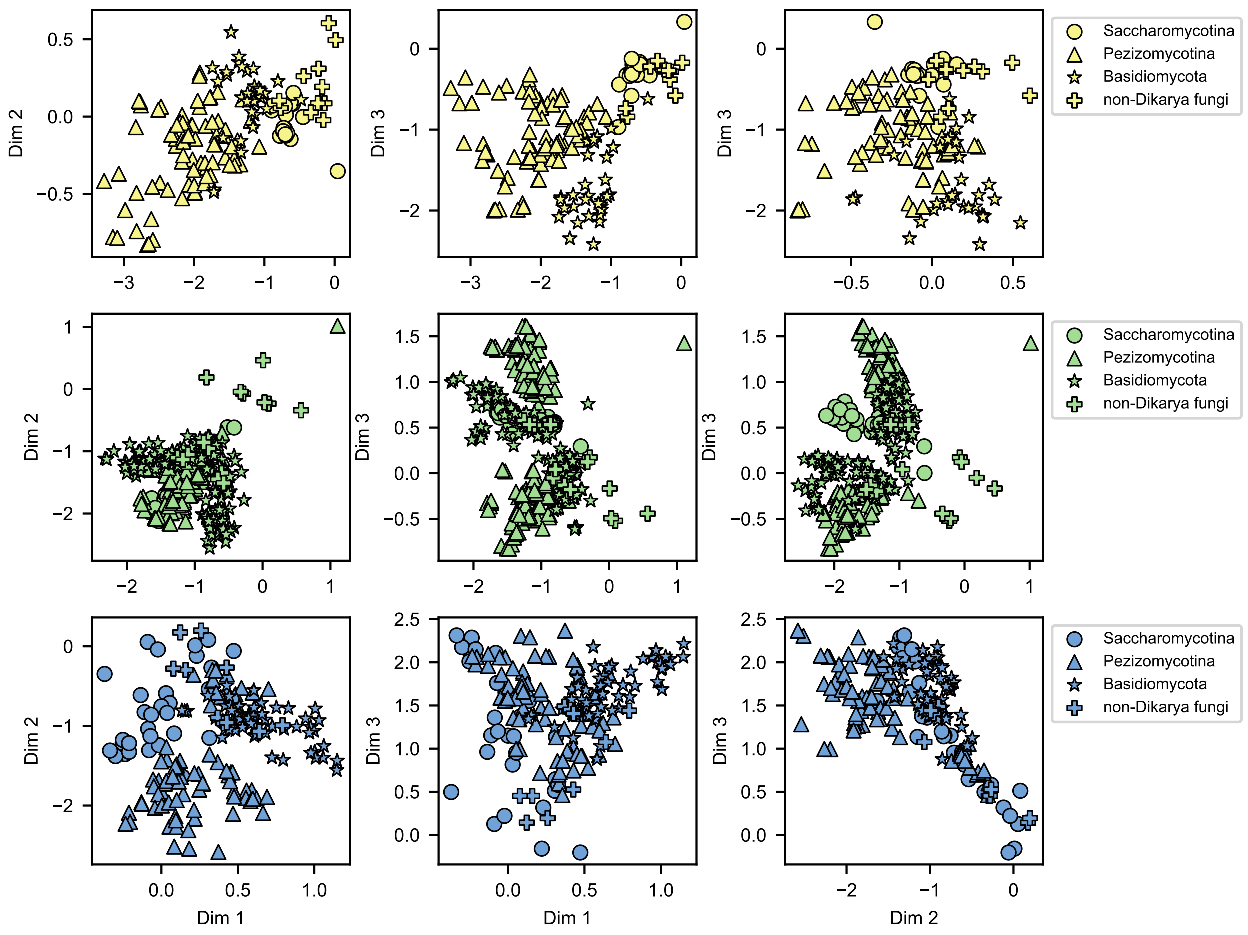


Figure 6: **InfoVAE latent space embeddings and phylogenetic annotation of the three additional SH3 paralog groups.**
Color schemes: Bzz11 (yellow), Abp1 (green), and Rvs167 (blue), showing the nested hierarchical organization by function (color) and phylogeny (symbol).

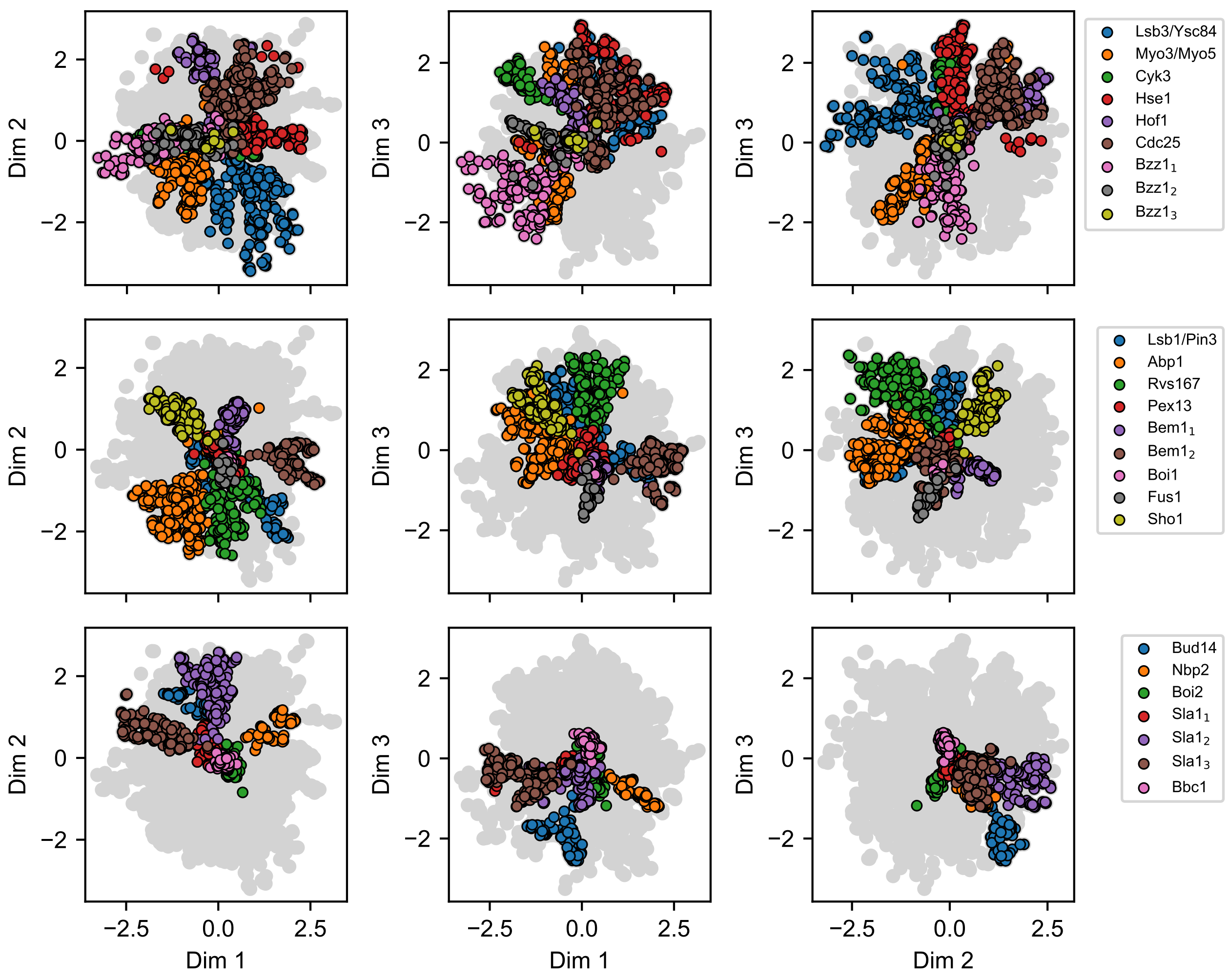


Figure 7: **InfoVAE latent space embeddings of all annotated SH3 paralog groups.**

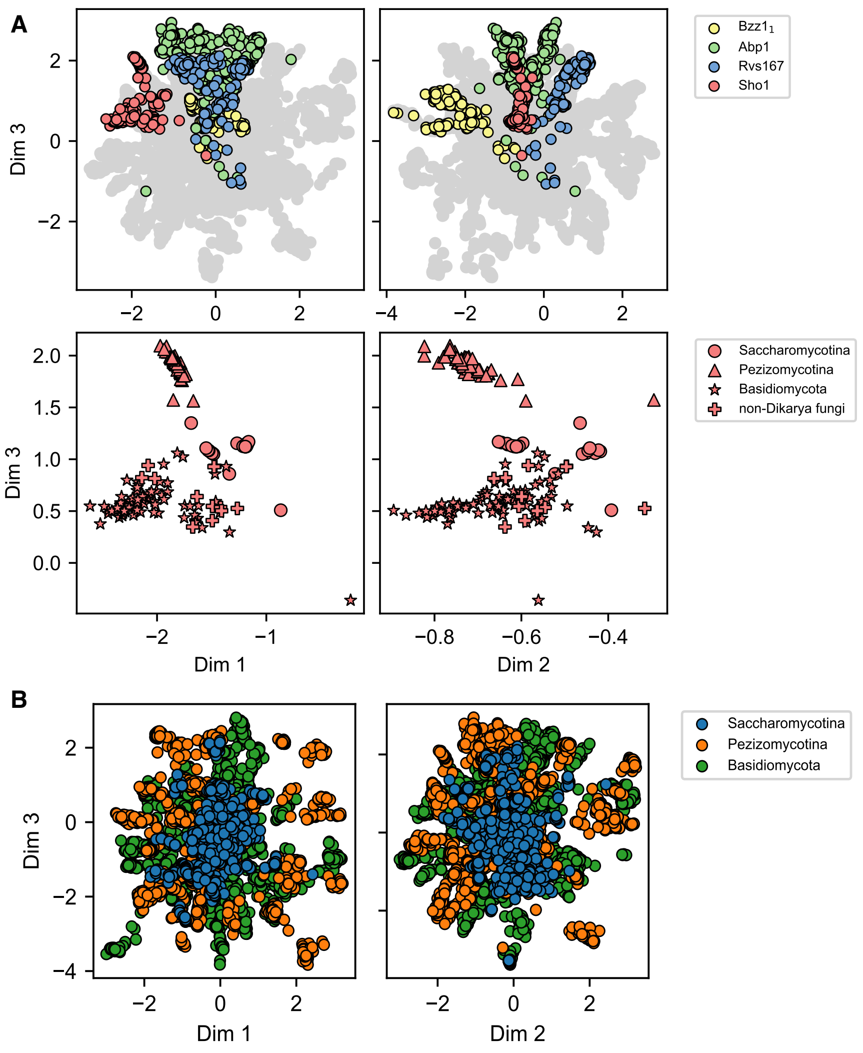
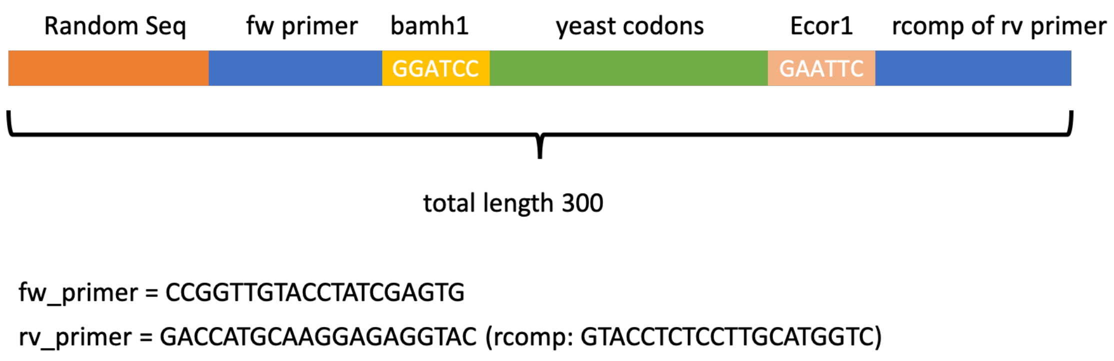


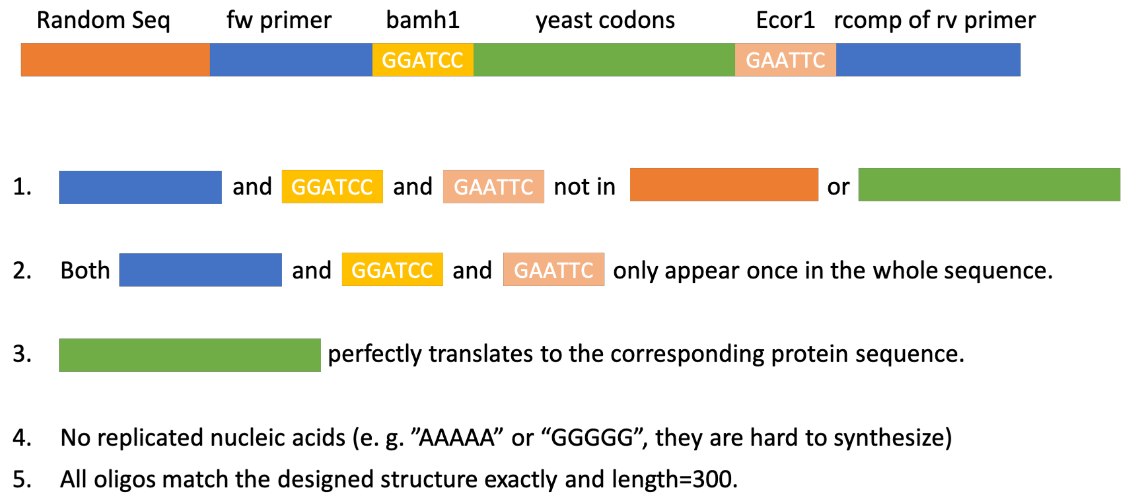
Figure 8: **The latent space of the vanilla VAE learns a nested hierarchical partitioning of natural fungal SH3 homologs by function and phylogeny.** (A) Annotation by paralog group and phylogenetic annotation within the Sho1 paralog cluster (red). (B) The vanilla 3D latent space embedding of the 5299 natural SH3 homologs annotated by the three main fungal phylogeny groups.



fw_primer = CCGGTTGTACCTATCGAGTG

rv_primer = GACCATGCAAGGAGAGGTAC (rcomp: GTACCTCTCCTTGCATGGTC)

Figure 9: **Structure of the oligo nucleotide sequences containing SH3 domains and adaptors.**



1. 🟦 and GGATCC and GAATTC not in 🟧 or 🟩
2. Both 🟦 and GGATCC and GAATTC only appear once in the whole sequence.
3. 🟩 perfectly translates to the corresponding protein sequence.
4. No replicated nucleic acids (e. g. ”AAAAA” or “GGGGG”, they are hard to synthesize)
5. All oligos match the designed structure exactly and length=300.

Figure 10: **Guidelines for verifying the designed oligo nucleotide sequences.** Steps of sequence checking to ensure accuracy, specificity, and compatibility with experimental conditions.

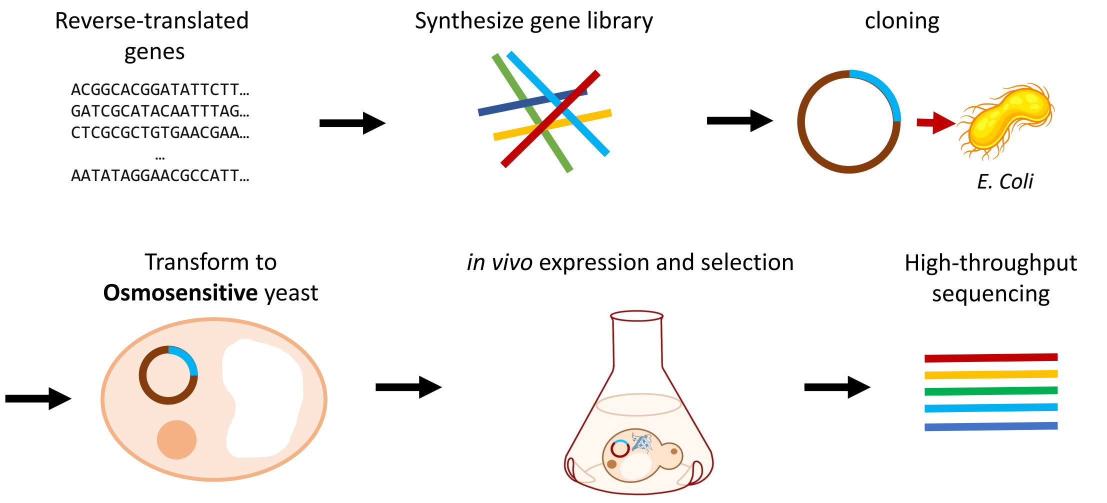


Figure 11: **Full experimental workflow for the yeast osmosensing assay**

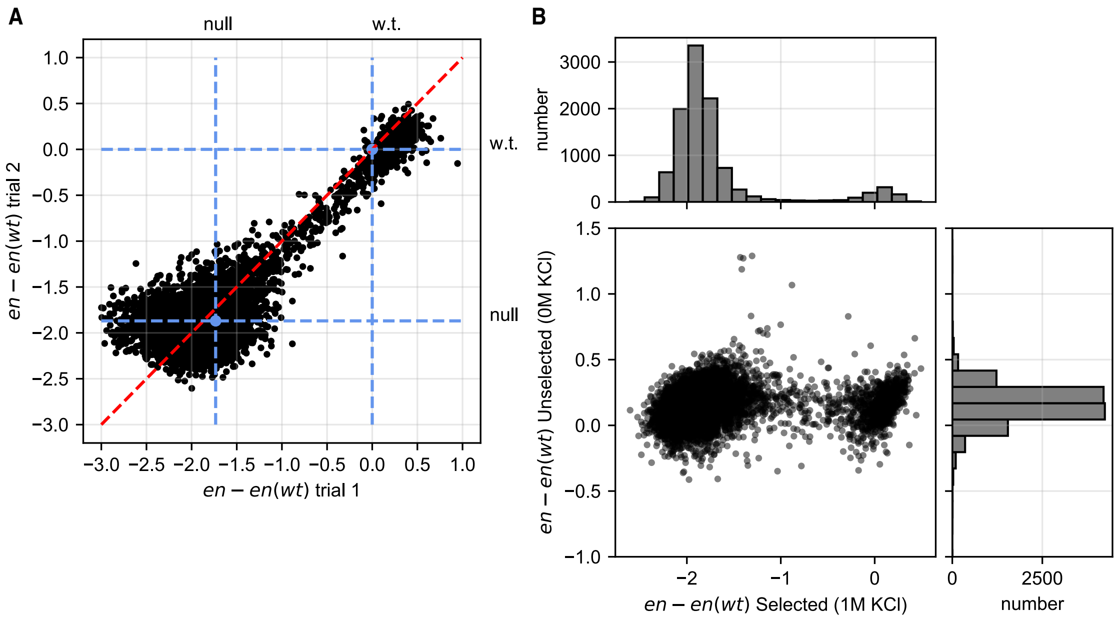


Figure 12: **Validation of the high-throughput select-seq assay.** (A) A scatterplot of the enrichment score relative to wild-type $[en - en(wt)]$ for two independent ($n = 11{,}442$) trials of the select-seq assay under the same experimental conditions. The position of the wild-type Sho1 sequence and the null allele (no Sho1 activity) are indicated by the blue circle and blue dashed lines. The red dashed line is the identity trace. Values at low values of $[en - en(wt)]$ are subject to more variability as expected from poorer counting statistics. The data show that the select-seq assay shows good reproducibility between independent runs ($\rho_{\text{Pearson}} = 0.87$, $n = 11{,}442$, $p < 1 \times 10^{-307}$). (B) The relationship between $en - en(wt)$ of the SH3 genes in *S. Cerevisiae* grown in selective (1M KCl) and non-selective (0M KCl) media. No statistically significant correlation is observed ($\rho_{\text{Pearson}} = 0.10$, $n = 10{,}448$, $p = 6 \times 10^{-23}$). This control experiment shows that bimodal distribution of enrichment in the selected population resulted from differential adaptability under high osmotic pressure conditions.

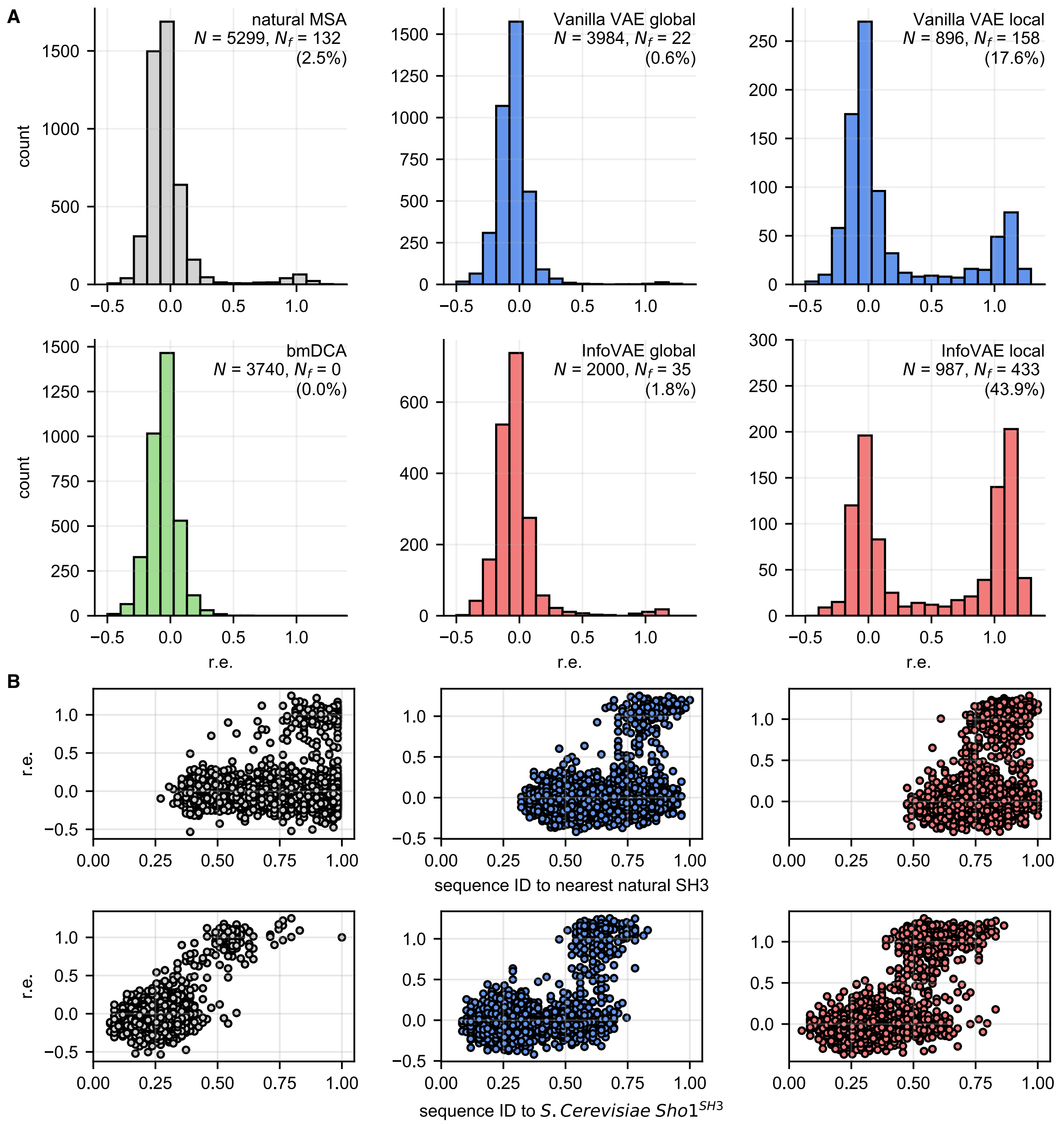


Figure 13: **Function and diversity of natural and synthetic SH3 variants.** (A-F) Distribution of r.e. scores measured by high-throughput select-seq assay for the 5299 natural SH3 homologs (A), 3740 bmDCA synthetic variants (B), 3984 global (C) and 896 local (D) vanilla VAE synthetic variants, and 2000 global (E) and 987 local (F) InfoVAE synthetic variants. (G-I) Scatterplots of r.e. vs. sequence identity (ID) to the nearest natural homolog or *S. Cerevisiae* Sho1$^{\text{SH3}}$ for the 5299 natural sequences (G), 4880 global and local vanilla VAE synthetic sequences (H) and 2987 global and local InfoVAE synthetic sequences (I).

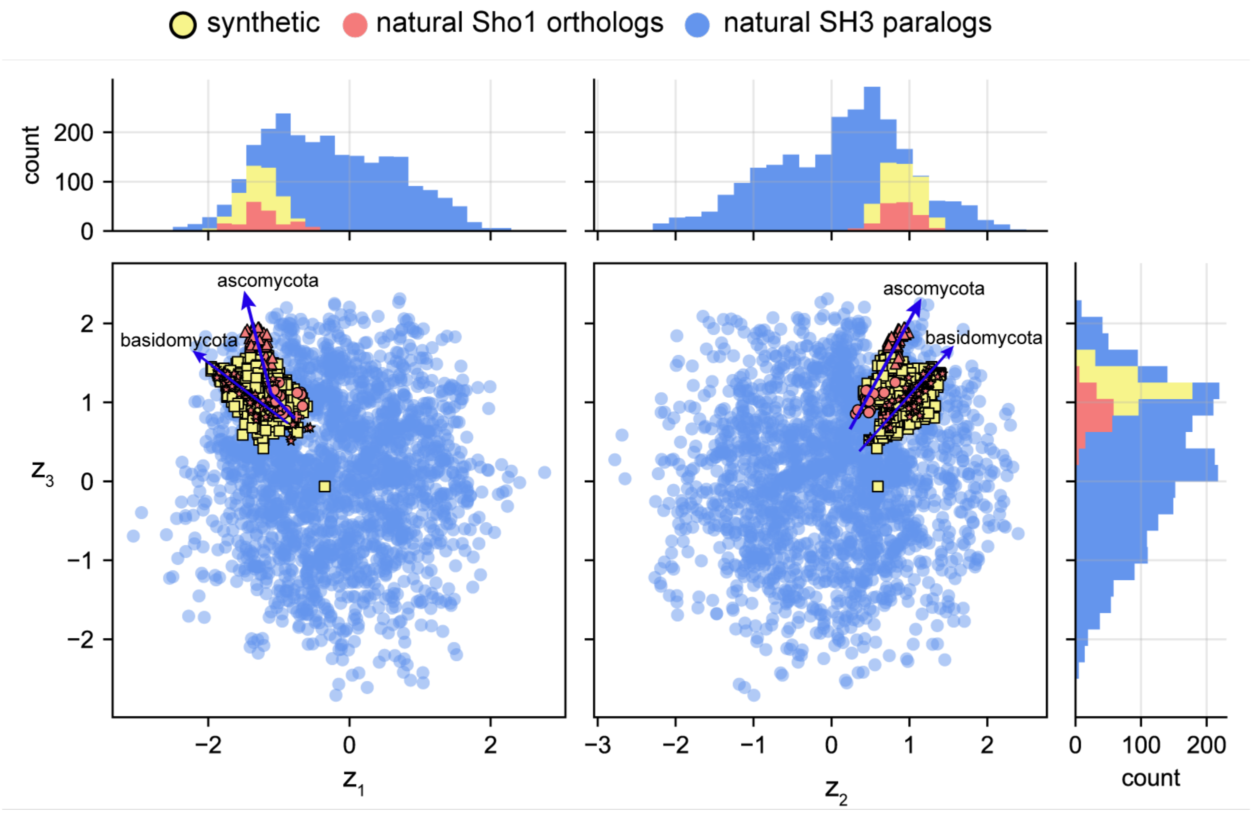


Figure 14: **Natural and synthetic functional Sho1$^{\mathbf{SH3}}$ orthologs in the VAE latent space**

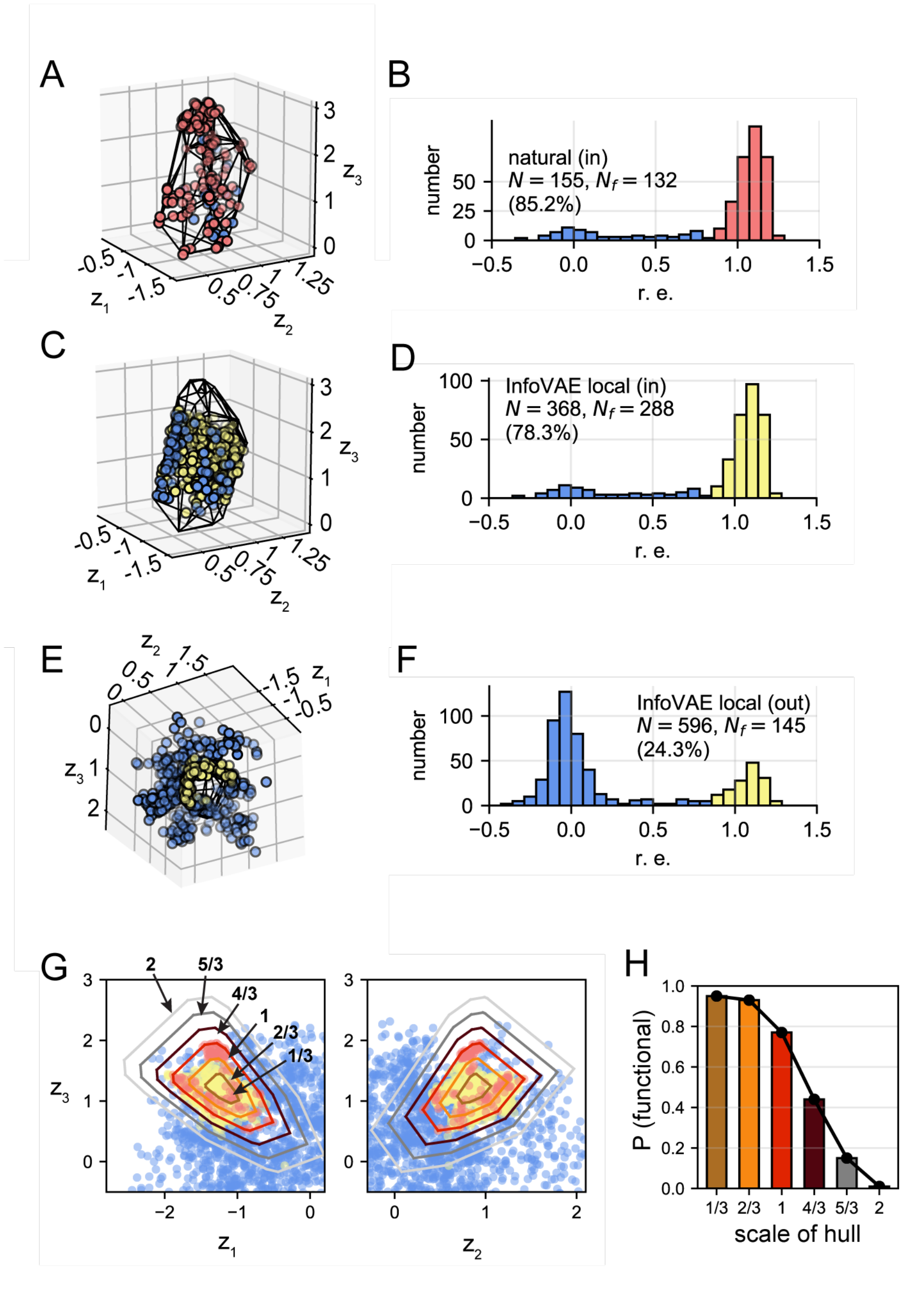
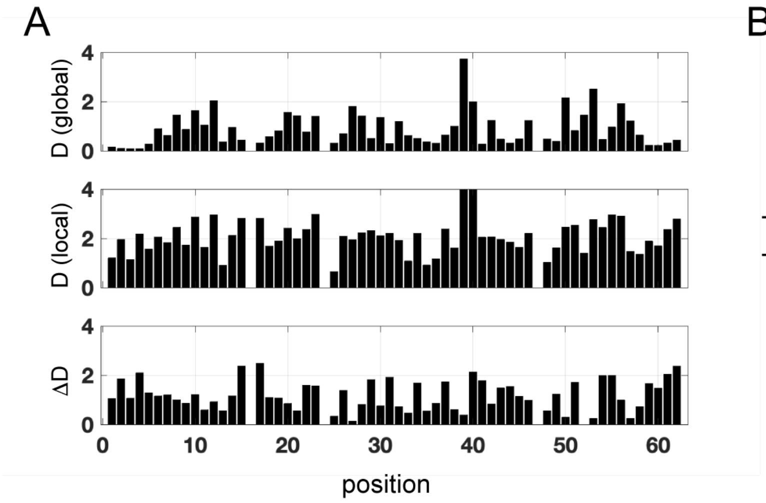
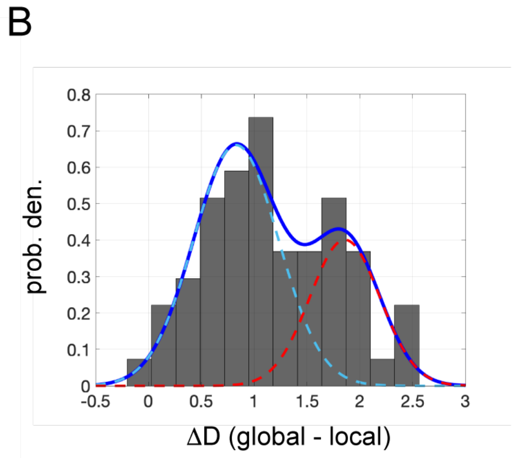


Figure 15: **Spatial localization of Sho1$^{\mathbf{SH3}}$ function in the VAE latent space.**

Figure 15

(A-B) Convex hull (black lines) of the natural functional SH3 orthologs (red) defined as the smallest convex polygon that encloses 132 functional SH3 homologs. A small number of 23 non-functional natural sequences (blue) are contained within the convex hull construction. The preponderance 85.2% of sequences contained within the convex hull are functional, indicating that localization within the region of latent space defined by the convex hull is a good proxy for osmosensing function. (C-D) Analysis of the synthetic sequences locally designed by the InfoVAE lying *within* the natural convex hull reveals 288 functional (yellow) and 80 non-functional (blue) synthetic variants, indicating that 78.3% of synthetic InfoVAE variants residing within the convex hull are functional. (E-F) Analysis of locally designed InfoVAE synthetic sequences lying *outside* the natural convex hull reveals 145 functional (yellow) and 451 non-functional (blue) synthetic variants, indicating that 24.3% of local InfoVAE variants residing in the vicinity of the convex hull are functional. (G) Illustration of the hulls scaled by 1/3, 2/3, 1, 4/3, 5/3, and 2 within 2D projections of the InfoVAE latent space and superposed upon the 132 functional natural SH3 orthologs (red), 468 functional synthetic proteins, and the rest of non-functional synthetic proteins (blue) generated by the InfoVAE. (H) Probability (P) of functional natural and InfoVAE designed sequences contained within each hull as a function of scaling factor.





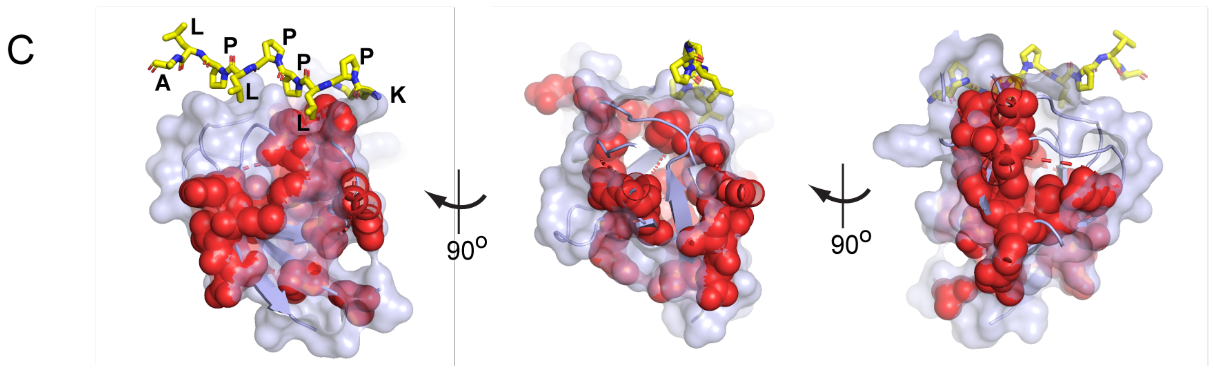


Figure 16: **The structural basis for Sho1$^{\mathrm{SH3}}$ function.**
(A) Positional conservation (measured by Kullback-Leibler relative entropy D) in sequences sampled globally from the InfoVAE latent space (top panel),locally from the convex hull bounding functional natural sequences (middle panel), and the difference of the two (bottom panel). This analysis exposes the extra constraints in SH3 domains to be specifically functional in the Sho1 osmosensing pathway. (B) The distribution of differences in conservation, with a fit to a double Gaussian mixture model (blue). For illustrative puproses, the mixture model helps to identify a population of 21 positions showing the largest change in conservation (red curve). (C) The positions showing the largest change in conservation (red speheres) are located at specificity determining regions of the ligand binding pocket and extending throughout the tertiary structure. The imgages show three rotations of the Sho1$^{\mathrm{SH3}}$ structure, with the co-crystallized Pbs2 peptide ligand in yellow stick bonds.

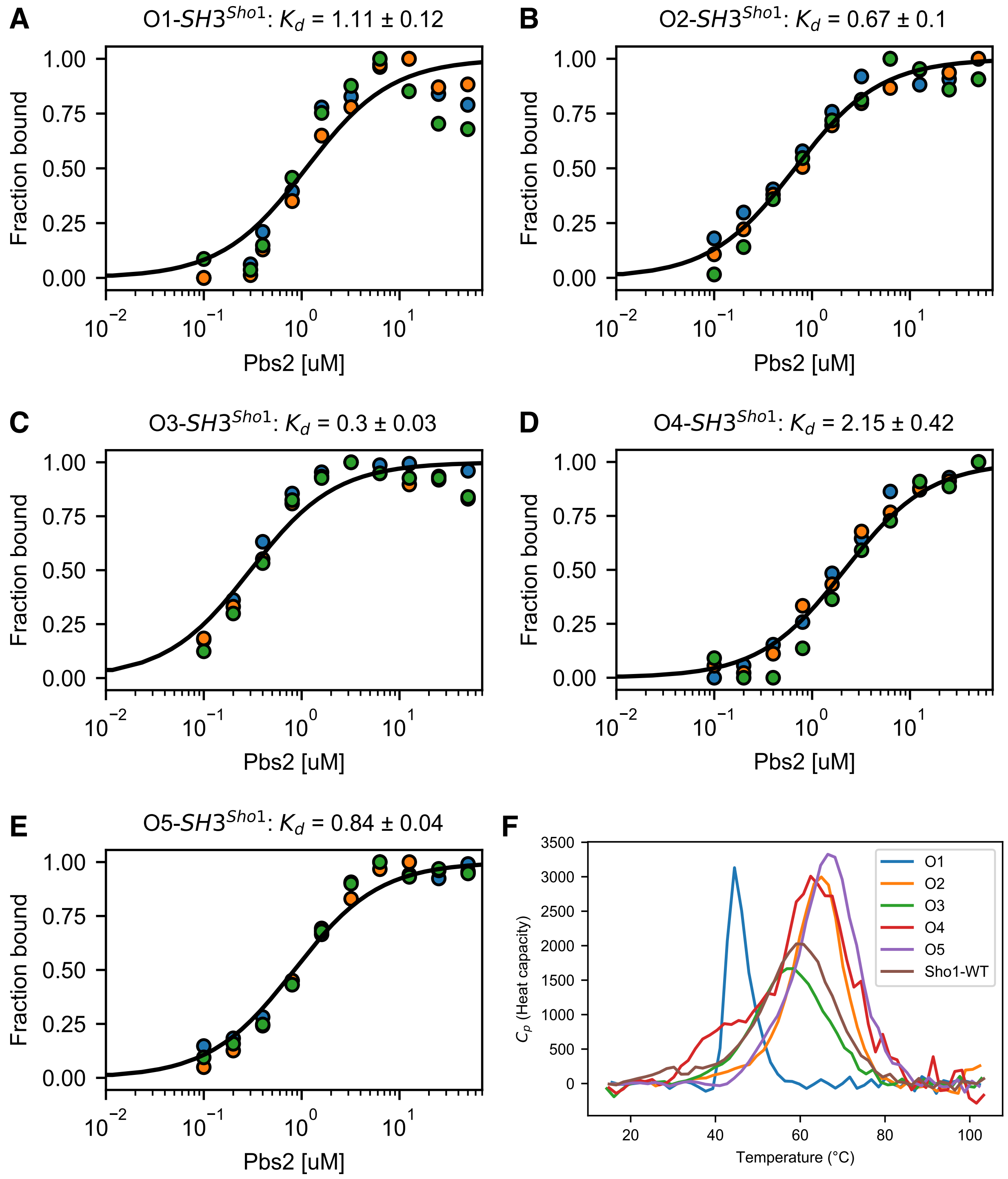


Figure 17: **Fluorescence titration curves of the five designed functional Sho1-SH3 orthologs listed in Table 1.**
(A-E) Titration curves of designed sequences InfoVAE_1 (O1), InfoVAE_2 (O2), InfoVAE_6 (O3), InfoVAE_10 (O4) and InfoVAE_11 (O5). (F) Melting temperatures were measured for the five designed sequences by differential scanning calorimetry (DSC).

LIST OF TABLES

Table 1: **Sequences of the five synthetic InfoVAE synthetic SH3 variants that rescue osmosensing function selected for purification and *in vitro* biophysical evaluation of ligand binding and folding**

Header	**Sequence**
InfoVAE_local_1	EYPYRAKAIYSYEADPDDANEISFTKHEILEISDVSGRWWQAKKADGTIGIAPSNYLILL
InfoVAE_local_2	DYAYKARALYAYTADDDDPNELSFAKGEVLDIVDNSGKWWQARKADGRTGIVPSNYMQLL
InfoVAE_local_6	PPAIKAKALYAYTADDDDPNELSFAKGEILDILDKSGKWWEARKADGSTGIAPSNYLQLV
InfoVAE_local_10	EYPYRAKAIYSYEADDDDANEISFTKGEILEISDVQGRWWQAKKADGTIGIAPSNYLQLL
InfoVAE_local_11	EYPYRAKALYSYQANPDDANEISFAKGEVLDISDVSGRWWQARKANGETGIAPSNYLQLL

Table 2: **Five synthetic InfoVAE synthetic SH3 variants that rescue osmosensing function plus wild-type *S.cerevisiae* Sho1$^{\mathbf{SH3}}$ selected for purification and *in vitro* biophysical evaluation of ligand binding and folding**

ID (WT) = sequence identity to wild-type Sho1^{SH3} [Marles et al., 2004],
ID (closest) = sequence identity to nearest natural SH3 homolog,
K_d = pbs2 MAPKK ligand dissociation constant measured by titration,
T_m = melting temperature measured by DSC,
ΔH = enthalpy of folding from two-state fit to DSC data.

Header	**Closest Sho1$^{\mathbf{SH3}}$ ortholog**	**ID (WT)**	**ID (closest)**	K_d **[μM]**	T_m **[°C]**	ΔH **[kJ/mol]**
WT	*Saccharomyces cerevisiae*	1.00	1.00	3.0±0.1	59.1	41.2 ± 0.3
InfoVAE_local_1	*Trichophyton rubrum*	0.53	0.92	1.1±0.1	44.5	41.5 ± 1.9
InfoVAE_local_2	*Moesziomyces antarcticus*	0.53	0.90	0.7±0.1	65.0	50.9 ± 0.7
InfoVAE_local_6	*Fistulina hepatica*	0.54	0.83	0.3±0.03	58.5	38.0 ± 0.3
InfoVAE_local_10	*Trichophyton rubrum*	0.56	0.85	2.2±0.4	62.5	41.6 ± 1.1
InfoVAE_local_11	*Neurospora crassa*	0.59	0.88	0.8±0.04	66.5	56.3 ± 0.6

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my supervisor, Rama Ranganathan, for his outstanding guidance and expertise in the wet lab research in biology. His vast knowledge in protein design, his exceptional mentoring, and his endless support have been critical to the success of my research. I feel fortunate to have worked with such a dedicated and brilliant mentor and I am indebted to him for his constant encouragement and support.

I would also like to acknowledge my co-supervisor, Andrew Ferguson, for his exceptional guidance in the computational research aspect of my project. His expertise in deep learning, his insightful feedback, and his unwavering support have been instrumental in my work. I am grateful for his mentorship and the opportunity to work with him.

I would also like to extend my gratitude to Niksa, an expert in deep learning, for his collaboration and contribution to this project. His expertise and insights in this field have been invaluable to the success of this research.

Lastly, I would like to thank all the collaborators and colleagues who have supported me during my Ph.D. journey. Their feedback, encouragement, and advice have been essential to my growth as a researcher, and I am deeply grateful for their help and support.

ABSTRACT

Protein design has emerged as an important field in contemporary biology, driven in part by the accumulation of vast amounts of protein data in public databases like the Protein Data Bank (PDB). The challenge now is to use this data to decipher the principles underlying protein design, as guided by nature, and to develop novel proteins with desired properties. To this end, we investigated the design principles of orthologs and paralogs of a small binding protein - Sho1^{SH3} - in the yeast osmosensing pathway. Using this natural system as a template, we employed deep learning models to design novel functional osmosensing orthologs. Our results demonstrate that these models not only accurately captured the distribution of functionality of natural proteins, but also expanded the functional space by designing novel proteins that extended beyond the functional constraints of natural proteins. This work provides valuable insights into the principles governing protein design and opens up new avenues for the development of novel proteins with desirable functions.

CHAPTER 1

INTRODUCTION

1.1 Exploring Protein Function and Sequence Space

Proteins are fundamental building blocks of living organisms, performing a wide variety of functions that are essential for life. The design of novel proteins with desired properties has become a crucial challenge in contemporary biology. With advances in technology, large amounts of protein data have been accumulated in public databases, it is now possible to mine this data and decipher the principles underlying protein design, as guided by nature, in order to develop novel proteins with specific functions.

Proteins with a wide range of properties and functions can potentially be designed and synthesized, opening up new avenues in biology and medicine. One particularly interesting protein module, the SH3 (Src Homology 3) domain [Musacchio et al., 1992] is a small binding protein found in many organisms, including yeast. It is involved in many cellular processes, including signal transduction and cell division, and has been extensively studied as a model system for protein design. In yeast, Sho1$^{\mathrm{SH3}}$ is part of the osmosensing pathway (Fig. 1A), which is responsible for the regulation of cellular responses to changes in osmotic pressure [Zarrinpar et al., 2003]. The orthologs and paralogs of Sho1$^{\mathrm{SH3}}$ in this pathway provide an excellent opportunity to investigate the principles of protein design and to design novel proteins with desired osmosensing functions.

In this study, we examined the design principles of orthologs and paralogs of SH3 domains in the yeast osmosensing pathway. We then used deep learning models to design novel functional osmosensing orthologs. Our results demonstrate that these models not only accurately captured the distribution of functionality of natural proteins, but also expanded the functional space by designing novel proteins that extended beyond the functional constraints of natural proteins. This work provides valuable insights into the principles governing protein design and opens up new avenues for the development of novel proteins with desirable functions.

Overall, this study highlights the importance of using natural systems as templates for protein design and the potential of deep learning models to expand the functional space of natural proteins. These insights provide a foundation for future research in protein design and have implications for the development of novel proteins with desired properties.

1.2 Diversity and Specificity of the SH3 Protein Family in Protein-Protein Interactions and Cellular Signaling

SH3 domains play a crucial role in protein-protein interactions and signaling pathways. They are present in a wide variety of proteins with diverse functions in different paralogous families. For example, in humans, SH3 domains are present in proteins such as Src and Grb2, which are involved in cell signaling and regulation.

SH3 domains are small all-beta folds that bind to type II poly-proline containing peptides of the form N-R/KXXPXXP-C or N-XPXXPXR/K-C [Musacchio et al., 1992](Fig. 1B) and mediate diverse signaling functions in cells [Mayer, 2001]. For example, a C-terminal SH3 domain in the Sho1 transmembrane receptor in fungi (Sho1^{SH3}) mediates the response to external osmotic stress through binding to a polyproline ligand in the Pbs2 MAP kinase (Fig. 1A). The Sho1 pathway has been conserved within the fungal kingdom through many speciation events, creating a diverse ensemble of extant Sho1^{SH3} ortholog sequences. In addition, duplication events have occurred during natural evolution, creating many paralogous SH3 domains that have diverged to acquire distinct and non-overlapping ligand specificities. For example, in *S. cerevisiae*, the Sho1^{SH3} is the only SH3 domain amongst 26 other paralogous domains in genome that can support osmosensing in the Sho1 pathway [Zarrinpar et al., 2003]. This exclusivity *in vivo* is recapitulated in direct binding assays with the Pbs2 ligand, demonstrating that the specificity is directly encoded in the Sho1^{SH3} amino acid sequence.

However, when move out from *S. cerevisiae* to a wider range of species, the situation appears to change. Of 12 metazoan SH3 domains tested by Zarrinapar et al, six reconstituted osmo-resistance when swapped into Sho1. Notably, these same six domains also exhibited binding to the Pbs2 ligand *in vitro*, as demonstrated through SH3 domain arrays and in solution binding assays using the free Pbs2 peptide (Fig. 1C).

The conservation and divergence of SH3 domains in different species have significant implications for their functional roles in mediating osmotic stress response and binding to the Pbs2 ligand. While the Sho1$^{\mathrm{SH3}}$ domain in *S. cerevisiae* demonstrates exclusive osmosensing capabilities among the 26 paralogous SH3 domains in its genome, the situation changes when considering a broader range of species. This observation suggests a potential variation in the specificity and functionality of SH3 domains across different species. Understanding the underlying factors governing the conservation and divergence of SH3 domains is critical for unraveling the intricate mechanisms of protein-protein interactions and signaling pathways in diverse organisms.

1.3 Date-driven Protein Design

An emerging approach for understanding and designing synthetic proteins is learning the design principles of natural proteins evolved through variation and natural selection. These principles are encoded within ensembles of homologous amino acid sequences and define the mapping from primary sequence to multifaceted protein phenotypes, including foldability, biochemical activities, and organismal fitness in a natural biological context [Anfinsen, 1973, Bowie et al., 1990, Socolich et al., 2005, Russ et al., 2005, 2020]. Evolution-based algorithms that learn these rules have the potential to generate new hypotheses for protein mechanism, and to permit the design of diverse synthetic variants with novel functions, with powerful implications for medicine, biotechnology, chemical engineering, and public health [Ferguson and Ranganathan, 2021].

Historically, protein design typically involve physics-based scoring functions that adopt tertiary structure as the central object to bridge sequence to function [Huang et al., 2016, Kiss et al., 2013, Anand et al., 2022] or involve directed evolution to learn a sequence to function mapping through iterative rounds of mutation and functional selection [Arnold, 2018, Jäckel et al., 2008, Romero and Arnold, 2009]. In recent years, advances in deep machine learning have driven exciting developments in machine learning-assisted directed evolution (MLDE) [Ferguson and Ranganathan, 2021, Freschlin et al., 2022, Bepler and Berger, 2021, Mazurenko et al., 2019, Wittmann et al., 2021, Frappier and Keating, 2021] that train models to learn the sequence to function map. The central idea of these strategies is to replace a blind mutational search through the vast gulf of protein sequence space with a model-guided search, and to eliminate the need for the direct use of structural information by implicitly representing the underlying physics in the model-learned parameters. The learned models provide a new understanding of the organizing principles of natural proteins at both in terms of general "linguistic rules" underpinning the patterns amino acids in all natural proteins and the local and global epistatic interactions between amino acids in individual proteins that provide for protein phenotypes [Halabi et al., 2009, Rivoire et al., 2016, Russ et al., 2020, Morcos et al., 2011, Ferguson et al., 2013, Hart and Ferguson, 2015, Mann et al., 2014, Hopf et al., 2017, Ding et al., 2019].

CHAPTER 2
YEAST OSMOSENSOR SHO1-SH3 – A MODEL SYSTEM FOR PROTEIN FUNCTION

Due to extensive past work documenting tight functional specificity *in vivo* and great functional diversity [Zarrinpar et al., 2003, Saksela and Permi, 2012], the SH3 domain family serves as a productive model system for studying the generative potential of data-driven models. In this chapter, we provide an overview of our experimental model system, which includes both *in vivo* and *in vitro* SH3 binding experiments. We specifically focus on our simple and efficient high-throughput yeast osmosensing experimental protocol (11), which allows for rapid validation of data-driven protein design models.

2.1 High-throughput Osmosensing Assay

2.1.1 Gene construction

Before gene construction, 1-3 positions of a small number of designed sequences were hand-adjusted to correct effect of misalignment in the training data. Residues 16D, 17D and 46A of Sho1 (PDB 2VKN) were inserted into each designed sequence to make a final length of 62. To avoid over-similarity, sequence samples were successively picked and filtered to maintain at least 3 amino acids distance away from any other candidates in each sample set.

S. cerevisiae codon-optimized genes coding (codes for reverse translation can be found at `https://github.com/ranganathanlab/Reverse_translation` and section 7.3) for all synthetic SH3 proteins were amplified from a mixed pool of oligonucleotide fragments synthesized on microarray chips (Twist). The oligonucleotides corresponding to each gene were designed with primer annealing sites and a padding sequence to make them uniform 300-mer (Fig. 9 and 10). PCR was performed using KAPA-Hifi polymerase with 1X KAPA HiFi Buffer (Roche), 0.2 mM dNTPs and 1.0 μm of each forward (5'-CCGGTTGTACCTATCGAGTG-3') and reverse primer (5'-GACCATGCAAGGAGAGGTAC-3') in 25 μl total volume, with an initial activation (95°C, 2 min), followed by 14 cycles of denaturation (95°C, 20 s), annealing (65°C, 10 s) and primer extension (70°C, 10 s). A final extension step (70°C, 2 min) was performed subsequently. Amplified products were column purified (Zymo Research), digested with EcoR1 and BamH1, ligated into the digested PRS136 plasmid with N-terminal membrane domain of Sho1 [Zarrinpar et al., 2003], and transformed into Agilent Electrocompetent XL1-Blues to yield >250× transformants per gene. The entire transformation was cultured in 50 ml LB media containing 100 μg/ml sodium ampicillin (Amp) at 37°C overnight after which plasmids were purified and pooled.

2.1.2 Yeast transformation

The haploid *S. cerevisiae* strain SS101 was constructed on the W303 background gifted by Wendell Lim (UCSF) [Zarrinpar et al., 2003]. Genetic knockouts of *Ssk2* and *Ssk22* were created to remove the Sho1-independent branch of the osmoresponse pathway [Posas and Saito, 1997]. The pooled pRS316 plasmids with the SH3 gene library were transformed into SS101 cells using the LiAc-PEG high efficiency transformation protocol [Gietz and Schiestl, 2007]. Plate check was performed to confirm at least 50 copies of each gene were successfully transformed. Transformed SS101 cells were grown in liquid Sc-Ura media for 24 h (add 20 mL Sc-Ura media for each 10^8 total transformed cells) at 30°C, and then passaged to 250 mL fresh liquid Sc-Ura media to make OD = 0.05. After another 24 h of growth at 30°C, the Sc-Ura culture can be kept at 4°C for up to two weeks. It is feasible to "accumulate" transformation efficiency by conducting 2-3 rounds of transformations on separate days, enabling the testing of approximately 12,000 SH3 sequences totally in one trial.

2.1.3 SH3 domain selection assay

All growth was at 30°C on shaker. The stock Sc-Ura culture was transferred to YPD media for a 24 h growth to get the t_0 sample. The culture was diluted every 8 h to keep the cell density below 0.2 OD_{600}. A small volume of the t_0 sample was transferred to YPD media supplemented with either (1) no KCl (non-selective) or (2) 1M KCl (selective), and the rest was span down and minipreped to extract plasmids from yeast. Both non-selective and selective cultures were grew for 24 h with OD_{600} maintained under 0.2 to obtain the t_{24} samples. The two t_{24} samples were span down and minipreped using the same protocol as the t_0 sample.

Plasmids purified from both t_0 and t_{24} samples were amplified using two rounds of PCR with Q5 polymerase (New England Biolabs) to add adapters and indices for Illumina sequencing. In the first round the DNA was amplified using primers that add from 6 to 9 random bases (Ns) for initial focusing, as well as part of the i5 or i7 Illumina adapters. Six cycles were used to minimize amplification-induced bias, followed by ampure purification before the second round PCR. In the second round of PCR, the remaining adapter sequence and TruSeq indices were added, where 20 cycles were used. The final products were gel purified (Zymo Research) , quantified using Qubit (ThermoFisher) and sequenced in an Illumina MiSeq system with a paired-end 300 cycle kit. Allele counts were obtained using standard procedures. Paired-end reads were joined using FLASH, trimmed to the EcoR1 and BamH1 cloning sites and translated. Only exact matches to the designed genes were counted. Enrichment (en) and relative enrichment ($r.e.$) values for each gene x of the three growth conditions were calculated according to equation:

$$en(x) = \log_{10}\left(\frac{f_{t24}^{x}}{f_{t0}^{x}}\right) \tag{2.1}$$

$$r.e.(x) = \frac{en(x) - en(null)}{en(wt) - en(null)} \tag{2.2}$$

where f^x_{t24} and f^x_{t0} represents the frequency of observing gene x in the t_{24} and t_0 sample, respectively. The wild-type sequence (wt) is the Sho1 gene of *S. cerevisiae* and the *null* genes are TAGNTAATTTCGGCGTGGGTATGGTGGCAGGCCCCGTGGCCGGGGGA CTGTTGGGCGCCATCTCCTTGCATGCACCATTCCTTGCGGCGGCGGTGCTCAA CGGCCTCAACCTACTACTGGGCTGCTTCCTAATGCAGGAGTCGCATAAGGGAG AGCGTCGAGAT, where the stop codon TAG produces a Sho1 without the C-terminal SH3 domain. A second independent selection assays was performed to ensure reproducibility (Fig. 12A). The average *en* of the two trials was used to calculate *r.e.*. r.e. values of SH3 variants with at least five counts in the input population in both trials were used for analysis. The r.e. scores between the natural and synthetic libraries were globally normalized to a single null allele by linear regression fitting over the 393 sequences shared between the two pools ($\rho_{\text{Pearson}} = 0.94$, $n = 393$). The standard curve relating r.e. to binding affinity was made from a set of 11 variants of *S. cerevisiae* Sho1$^{\text{SH3}}$, comprising D18I, R38L, Y56I, Y56F, Y56M, Y56A (from published data [Marles et al., 2004]), A8V, Y10M, P11D, E21G and wild-type.

2.2 *in vitro* SH3 Test Assays

2.2.1 Peptide synthesis

The pbs2 MAPKK peptides were synthesized with standard 9-fluorenylmethoxycarbonyl (Fmoc) chemistry in Protein Chemistry Technology Center of UT Southwestern Medical Center. Molecular masses were verified by mass spectrometry. Concentrations were verified by quantitative amino acid analysis.

2.2.2 Protein expression and purification

pET-28b plasmids encoding selected C-terminally His_6-tagged versions of functional SH3 domains were transformed into *E. coli* strain BL-21 (DE3). 1L of TB media containing 50 μg/mL kanamycin were inoculated 1:1000 with overnight starter LB cultures, grown in 37°C and 200 rpm to an OD_{600} of 0.8-1.2, induced with 200 μM IPTG and further incubated at 18°C overnight. Cells were harvested by centrifugation (2560 g, 15min) and resuspended in 500 mM NaCl, 10 mM imidazole, 25 mM Tris-HCl, pH 8.0 and 1:1000 Tween® 20 detergent, lysed by sonication on ice (3 rounds for each 100 mL cell suspension, 90% amplititude, 2s on 2s off for 1 min in total) with 1mM PMSF, 10 μg/mL leupeptin and 2 μg/mL pepstatin, and centrifuged at 48000 g for 1 h at 4°C. SH3 proteins were purified from the cleared lysate by Ni-NTA affinity chromatography (Qiagen) , dialyzed overnight in 100 mM NaCl, 50 mM Tris, pH 7.5, and run through size exclusion in fast protein liquid chromatography (AKTA Pure 25 L1). Purified SH3 protein can be flash frozen and stored at (-80)°C.

2.2.3 Biophysical evaluation of SH3 in vitro binding assay

Binding affinity to synthetic pbs2 MAPKK ligands for Sho1^{SH3} domains were measured by increase in the intrinsic tryptophan fluorescence on titration of peptide ligand into a solution of Sho1^{SH3} protein at a fixed concentration of 0.25 μM (less than one fourth of expected K_d) in HEPES buffer (20mM HEPES, 50mM NaCl, pH 7.3-7.6). Fluorescence titration was performed on Fluorolog-3 with $\lambda_{ex} = 296$ nm and $\lambda_{em} = 330$ nm. Data were fitted to the equation

$$y = F_{min} + (F_{max} - F_{min}) \left(\frac{x}{K_d + x} \right) \tag{2.3}$$

with scipy.optimize.curve_fit module in Python, where y is the fluorescence reading, x is ligand concentration, K_d is dissociation constant, F_{min} and F_{max} are minimum and maximum fluorescence values.

2.2.4 Melting temperature measurements

The melting temperature (T_m) of SH3 domains was determined using a MicroCal VP-Capillary DSC (Malvern Instruments). Purified protein (60 μM-180 μM) in 100mM NaCl, 50mM Tris·HCl, pH 7.5 was heated from 10°C-110°C at 60°C/hour and the resulting curve was fit using a two-state model.

CHAPTER 3
DATA-DRIVEN MODELS FOR DESIGNING NOVEL FUNCTIONAL OSMOSENSING ORTHOLOGS

Two MLDE approaches that have demonstrated particular promise are direct coupling analysis (DCA) and deep generative modeling (DGM). (Fig. 2A) These models may differ in their underlying principles, but share a common paradigm involving a compression step that maps a sequence alignment of natural homologs onto a low-dimensional parameter space, followed by a de-compression step that designs protein sequences from these parameters. In the osmosensing design task, The synthesized protein sequences are then subject to high-throughput gene synthesis and selection assays to evaluate their functionality as is described in 2.1.

3.1 Data Collection: Acquire the Natural SH3 Library

We assembled a comprehensive library consisting of SH3 domains extracted from a diverse array of genomes of 222 fungal species. This collection encompasses 5610 unique sequences, which includes, on average, 24 SH3 paralogs and one Sho1^{SH3} ortholog per genome. In addition, we incorporated a set of non-fungal SH3 domains, amounting to 2255 sequences. The total repertoire, therefore, stood at 7865 SH3 domains. The data for this extensive compilation were sourced from three major databases. The JGI Mycocosm database (`https://jgi.doe.gov`), which houses the data from the "1000 fungal genomes" project, the PFAM database (`https://pfam.xfam.org/`), and the NCBI non-redundant sequence database (`https://www.ncbi.nlm.nih.gov`), accessed through BLAST searches. We retained the natural nucleotide sequences of all SH3 domains and cloned them into an appropriate vector for subsequent expression and selection assay in *S. cerevisiae* as is described in section 2.1.

3.2 bmDCA

The essence of DCA is to start with a multiple sequence alignment (MSA) of a protein family and infer a generative model representing the intrinsic constraints on amino acids (the "one-body" terms) and the pairwise interactions between amino acids (the "two-body" terms) [Morcos et al., 2011, Cocco et al., 2018, Ferguson et al., 2013, Hopf et al., 2017, Tian et al., 2018]. For the chorismate mutase enzyme family, recent work showed that the DCA model is sufficient to design of synthetic variants that function in a manner equivalent to natural enzymes both *in vitro* and *in vivo*, in *E. coli* cells [Russ et al., 2020]. The relative simplicity of the constraints imposed by the DCA model led to considerable sequence divergence in the synthetic proteins, demonstrating access to an enormous space of functional proteins consistent with the evolutionary constraints.

The DCA model is relatively simple because it is inferred only from the first- and second-order statistics of sequence alignments. Given this, it is impressive that it can suffice to capture the design constraints for specifying proteins that can fold and function in their natural cellular context. However, it is also true that the chorismate mutases largely represent a family of orthologs - extant proteins that are descended by speciation events and are expected to share the same function across species. Indeed, a large fraction of homologous chorismate mutases operate in *E. coli* in the specific experimental conditions in which the design was carried out [Russ et al., 2020]. Such consistency of function in a protein family likely represents a simpler problem for inference of generative models. A deeper and more general test of evolution-based generative models would come from a study of a family of paralogs - proteins that arose through gene duplication events and typically have diverged to carry out distinct and specialized functions. Hence, we used Boltzmann machine direct-coupling analysis (bmDCA) [Cocco et al., 2018] for this test (Fig. 2B). The DCA approach assumes that the probability of each natural amino acid sequence $x = (x_1, \ldots, x_L)$ to occur is exponentially related to an "energy" function parameterized by the intrinsic constraints on each amino acid x_i at each position i ($h_i(x_i)$) and the pairwise couplings between amino acids (x_i, x_j) at positions (i, j) ($J_{ij}(x_i, x_j)$):

$$P(x) \propto \exp\left[\sum_i h_i(x_i) + \sum_{i<j} J_{ij}(x_i, x_j)\right] \tag{3.1}$$

The parameters (h, J) are trained to reproduce the empirical positional frequencies and pairwise correlations of amino acids (the one- and two-body statistics) in the input MSA. If the model accounts for the information content of natural sequences, synthetic sequences drawn from this probability distribution with low energy (that is, high probability) should be natural-like proteins. Boltzmann machine learning is computationally intensive but provides accurate fitting; for example, the trained bmDCA model for the SH3 family shows excellent reproduction of the input sequence statistics. As with any machine learning algorithm, bmDCA involves setting various parameters during model training. Here we follow the approach in previous work [Russ et al., 2020] to test whether the design of members of the ortholog family studied in that work generalizes to a functionally diverse family of paralogs. We generated synthetic sequences ($N = 3740$) that reproduce the same distribution of statistical energies (e.g. same probability) as the natural homologs (Fig. 3B) [Russ et al., 2020].

3.3 Variational Autoencoders (VAEs)

The second class of models we examined are DGMs known as a variational autoencoders (VAEs) [Kingma and Welling, 2013], consisting of two back-to-back deep neural networks: an encoder $q_\phi(z|x)$ that compresses the information content of sequences x in the MSA into low-dimensional latent space vectors z, and a decoder $p_\theta(x|z)$ that performs the reverse process, transforming latent vectors z back into protein sequences x (Fig. 2C). If the learning was effective, the latent space should reveal functional and/or evolutionary relationships between sequences, and the decoding process should generate novel sequences from latent space coordinates not occupied by natural sequences. The former operation can be thought of as an interpretive function of the VAE, while the latter represents novel design. In contrast to bmDCA, which learns on the one- and two-body amino acid statistics, the VAE models are trained to reconstruct all features of the input data, and make no assumptions about the form of the sequence-function model. This approach takes advantage of the powerful representational capacity of the deep neural networks [Chen and Chen, 1995, Hassoun, 1995], and provides a direct solution for designing novel sequences from the latent space without the need for computationally expensive numerical simulations [Hawkins-Hooker et al., 2021, Sinai et al., 2021, Dean and Walper, 2020, Giessel et al., 2022].

We implemented two forms of a VAE: (1) a generic, widely-used form that we call the "vanilla-VAE", and (2) a variant known as an information maximizing VAE (InfoVAE) [Zhao et al., 2019]. While the generic algorithms have proven useful for studying protein properties [Doersch, 2016, Guo et al., 2020, Greener et al., 2018, Riesselman et al., 2018, Sinai et al., 2017, Ding et al., 2019, Hawkins-Hooker et al., 2021, Dean and Walper, 2020, Sinai et al., 2021], they can also lead to inaccurate latent inference and non-optimal decoder performance [Sutskever et al., 2014, Rezende and Viola, 2018]. The InfoVAE addresses these problems, incorporating additional constraints during training models that encourages more accurate decoding from the latent space for design [Zhao et al., 2019]. We present data on both VAE architectures in this work, but for brevity, we illustrate features of the latent space representations in figures below using the infoVAE method.

For the SH3 designing task, we generated libraries of synthetic sequences from the latent space of both vanilla (N=3984) and infoMAX (N=2000) models by randomly sampling latent space coordinates and passing them through the decoder to convert into protein sequences. Re-embedding the designed sequences using the encoder demonstrates that they globally sample the latent space in both models (Fig. 4).

3.4 Experimental Evaluation

3.4.1 High-throughput Osmosensing Assay

Results of the yeast osmosensing assay shows no bmDCA designed sequences are capable of full complementation of the Sho1 deletion phenotype, though a few sequences fall into a partial rescue range (Fig. 13B). This result is particularly interesting since previous work by Best and colleagues [Tian et al., 2018] convincingly demonstrates that the bmDCA model is fully capable of producing well-folded and stable SH3 domains. Thus, it appears that bmDCA suffices to make folded SH3 proteins, but at least as tested here, does not capture enough information to specify orthologous function. This outcome could arise either from limitations imposed by using only pairwise statistics in the MSA or from the various approximations and parameter choices used in inferring the model [Kleeorin et al., 2021]. Regardless, the central conclusion is that at least for Sho1^{SH3}, simply reproducing the statistical energies of natural sequences in the bmDCA model is not sufficient to reproduce the distribution of function. In contrast, both VAE models are able to produce variants that rescue Sho1 function to the same level as wild-type *S. cerevisiae* Sho1^{SH3} (Fig. 13C, 13E), albeit with different yields. Specifically, 0.6% of vanilla-VAE and 1.75% of infoVAE designed sequences fully function in the Sho1 pathway. A two-sample Kolmogorov-Smirnov test shows that the vanilla-VAE distribution deviates from the natural distribution ($p = 1 \times 10^{-4}$), but that the InfoVAE distribution is statistically nearly the same ($p = 0.06$). These data show that both VAE models have the capabilities to design functional synthetic orthologs of *S. cerevisiae* Sho1^{SH3} but as expected, the InfoVAE model more accurately represents the design rules embedded in the natural ensemble.

The localization of natural Sho1^{SH3} orthologs in the latent space (Fig. 5B) suggests an additional hypothesis - that sampling in the immediate vicinity of natural orthologs should enrich the yield of synthetic orthologs. To test this, we computed the mean and variance of the functional natural orthologs and designed libraries of sequences from latent space coordinates sampled from the corresponding Gaussian distribution ($N = 896$ and $N = 987$ for vanilla- and info-VAE, respectively). A re-embedding of these sequences shows that they return to the environment from which they were sampled (Fig. 4), a quality check on the robustness of the VAE model in these regions. Experimental testing shows that indeed, local sampling produces a much higher density of fully functional synthetic orthologs (Fig. 13D, 13F). Thus, locality in latent space corresponds to locality the sequence-function mapping, even for models trained on sequence data alone and no prior knowledge of function.

What is the diversity of the new synthetic variants with respect to natural SH3 domains? For comparison, Fig. 13G shows the distribution of top sequence identities of natural sequences to their nearest natural counterpart or to *S. cerevisiae* Sho1$^{\mathrm{SH3}}$. Functional Sho1$^{\mathrm{SH3}}$ orthologs are more sequence similar to each other (>60% top-hit identity) than to SH3 paralogs, but can be quite diverged from *S. cerevisiae* Sho1$^{\mathrm{SH3}}$ (as low as 40% identity). The vanilla- and info-VAE methods approximate the same diversity, both in terms of distance from all Sho1$^{\mathrm{SH3}}$ orthologs and from the *S. cerevisiae* variant (Fig. 13H-I). The ability to reproduce the sequence diversity of natural homologs suggests that the models learn the physical constraints on orthologs without extensive overfitting on irrelevant idiosyncrasies of extant variants.

3.4.2 in vitro SH3 Test Assays

We selected five synthetic orthologs that show full function *in vivo* (1) for in-depth biochemical characterization. These proteins were expressed in *Escherichia coli* as His6-tagged fusions, purified to homogeneity, and assayed for (1) binding to the *S. cerevisiae* Pbs2 target peptide using a standard tryptophan fluorescence assay [Lim et al., 1994] and (2) thermal stability by differential scanning calorimetry. The data show that the synthetic proteins are well expressed, soluble, and display a range of binding affinities that are comparable to, or stronger than, the value for wild-type *S.cerevisiae* Sho1$^{\mathrm{SH3}}$ (Table 2, Fig. 17). Thermal denaturation experiments show that the synthetic proteins show cooperative unfolding transitions with half-maximal melting temperatures (T_m) and enthalpies of unfolding that span a range around the wild-type protein. Thus, the synthetic variants display biochemical properties similar to natural Sho1$^{\mathrm{SH3}}$ domains.

3.5 Methods

3.5.1 bmDCA

The 5299 natural sequences in MSA were used to infer the bmDCA model [Russ et al., 2020], assigning a probability $P(a_1, ..., a_L) = \frac{1}{Z}\exp\{-H(a_1, ..., a_L)/T\}$ to each aligned sequence $(a_1, ..., a_L)$ with $L = 59$. The statistical energy $H(a_1, ..., a_L) = -\sum_{1\leq i<j\leq L} J_{ij}(a_i, a_j) - \sum_{1\leq i\leq L} h_i(a_i)$ of the Potts model is given in terms of the direct coevolutionary coupling $J_{ij}(a, b)$ between amino acids a and b at positions i and j, and propensities $h_i(a)$ for the usage of amino acid a at position i. The bmDCA model was inferred at $\lambda = 0.01$ and M=500 using 1600 thermalization steps, and the temperature T is set to unity during inference. The accuracy of the inferred model was checked by comparing first order empirical frequencies f_i^a for each amino acid a at position i, and the joint frequencies $f_{i,j}^{ab}$ of amino acids (a, b) at positions (i, j), between the MSA and sequences generated by MCMC at $T = 1$. After the correspondence of the natural and predicted one- and two-body amino acid statistics were validated (Fig. 3A), the final bmDCA designed sequences for experiments were sampled under a lower temperature $T = 0.9$ to produce sequences that have compatible statistical energies with natural sequences (Fig. 3B) [Russ et al., 2020]. Codes for bmDCA are available at `https://github.com/ranganathanlab/bmDCA` [Barrat, 2020].

3.5.2 Vanilla VAE

Each natural homolog within the MSA was converted into a one-hot encoded tensor [Harris David, 2013], which maps each individual amino acid label found along a specific sequence into a vector consisting of zeros and ones, where the value 1 indicates the amino acid label. The unique labels consist of at maximum 20 amino acids and deletion gap from the multiple sequence alignment algorithm, and each amino acid position was indexed individually to avoid all-zero features. Thus, the MSA with size 5299×59 is converted into size 5299×1178, where the values 5299 and 1178 corresponds to the number of natural homologs used for training and length of the one hot encoded vectors. By using a training dataset that consists of homologs, a variational autoencoder (VAE) was employed as a generative model admitting a low-dimensional embedding of sequence space [Ferguson and Ranganathan, 2021, Sinai et al., 2017]. With the ability to capture meaningful SH3 evolutionary information through the latent space, this modeling approach has been attractive for protein design, while also introducing the opportunity to capture the distribution of this large homology family which can lead to better understanding of the evolutionary constraints for orthology and paralogy. For example, the decoder can sample from the latent space embedding and generate new artificial and functional protein sequences, while also localizing function within the latent space.

A standard "vanilla" VAE [Doersch, 2016, Guo et al., 2020, Greener et al., 2018, Riesselman et al., 2018, Sinai et al., 2017, Ding et al., 2019, Hawkins-Hooker et al., 2021, Dean and Walper, 2020, Sinai et al., 2021] was trained to learn the joint probability from Bayes inference: $p_\theta(x, z) = p(z)p_\theta(x|z) = p_\theta(x)p_\theta(z|x)$, where θ represents learned parameters of the joint distribution, $z \in Z$ represents latent variables, and $x \in X$ represents each sequence x in the training set X of MSA. $p_\theta(x, z)$ denotes the probability of correctly constructing a sequence like those in X given a z from the latent distribution $p(z)$. Each designed sequence $\hat{x}$ was generated by the decoder $p_\theta(\hat{x}|z)$ and z was sampled from $p(z)$. To learn $p_\theta(\hat{x}|z)$, we need to approximate $p_\theta(x)$:

$$p_\theta(x) = \int p_\theta(x|z)p(z)dz \tag{3.2}$$

Because it is intractable to directly compute parameters θ for the probability $p_\theta(x)$, we applied an approximation method called variational inference. Namely, we trained the encoder $q_\phi(z|x) \sim \mathcal{N}(z|\mu_\phi(X), \Sigma_\phi(X))$ parametered by ϕ, which takes values from X and outputs a multivariable Gaussian distribution over Z to approximate the posterior distribution $p_\theta(z|x)$ [Doersch, 2016, Ding et al., 2019], by minimizing the Kullback-Leibler divergence D_{KL} between $q_\phi(z|x)$ and the multivariable normal prior distribution $p(z) \sim \mathcal{N}(z|0,1)$. The logarithm of $p_\theta(x|z)$ term is approximated by the expectation $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$. Hereby, the VAE uses the loss function called Evidence Lower BOund ($\mathcal{L}_{ELBO}$) to maximize $\log p_\theta(x)$ [Doersch, 2016]:

$$\log p_\theta(x) \geq \mathcal{L}_{ELBO} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \tag{3.3}$$

where the encoder $q_\phi(z|x)$ is learned by taking X and optimizing ϕ, and the decoder $p_\theta(\hat{x}|z)$ is learned by taking Z and optimizing θ.

Both the encoder and decoder are implemented as fully connected feedforward artificial neural networks with three hidden layers (En1, En2, En3 for encoder and De1, De2, De3 for decoder). Two Dropout layers ($p = 0.7$) are between (En1, En2) and (De2, De3). Three Batchnorm layers are between (En2, En3); (De1, De2), and De3 and the output layer. The number of units in each hidden layer is 1.5 times length of the one-hot sequence. The activation functions between linear layers is tanh, while final decoder layer uses softmax neurons. We used PyTorch [Paszke et al., 2017] to implement our VAE model and trained our model using ADAM optimizer [Kingma and Ba, 2014] with a learning rate of 0.001. We used a 3D latent space based on results of five-fold cross validation [Kohavi, 1995] by taking into consideration both validation error and gap between training and validation error.

Training was conducted for 55 epochs where validation loss stopped decreasing. Codes for the vanilla VAE model are available at `https://github.com/ranganathanlab/VAEforDesign` and chapter 7.

Global sampling from the trained vanilla VAE model was conducted by randomly sampling 400 latent vectors from the Gaussian prior $p(z) \sim \mathcal{N}(0,1)$. We passed each latent vector z through the trained neural network decoder $p_\theta(x|z)$ to convert these into complete protein sequence with amino acid labels. Decoding requires multinomial sampling over the decoded probability distributions over the amino acids at each position in order to collapse the probability distribution into an unambiguous amino acid label. As such, we perform the decoding operation 10 times for each latent vector z to generate a total of 4000 globally-designed sequences. Local sampling was conducted by randomly sampling 150 latent vectors from the Gaussian prior $p(z) \sim \mathcal{N}(\mu_{top}, \Sigma_{top})$, where μ_{top} and Σ_{top} are mean and variance respectively of latent vectors of the high-r.e. natural homologs that rescue osmosensing function. For each vector, 10 sequences were generated by multinomial sampling from the decoded vector for a total of 1500 locally-designed sequences. The sequences were then filtered to eliminate highly similar sequence resulting in the production of 3984 globally-sampled sequences and 896 locally-sampled sequences.

3.5.3 InfoMax VAE

The implementation of InfoVAE is a collaborative effort with N. Praljak, as described in [Lian et al., 2022]. One limitation of Vanilla VAE is that optimizing the evidence lower bound objective (ELBO) is prone to learning a poor amortized inference distribution $q_\phi(\mathbf{z}|\mathbf{x})$ that may not closely approximate the true and expected posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ [Zhao et al., 2019]. There are two main reasons why these issues arise: (1) inherent properties of the ELBO objective and (2) implicit modeling bias. To overcome these issues, we defined a new training objective which learns a model to correctly reconstruct sequence and amortized inference distributions [Zhao et al., 2019]. First, we used an equivalent formation of the vanilla VAE ELBO objective:

$$\mathcal{L}_{ELBO} = -\mathcal{D}_{KL}\Big(q_\theta(z)\Big|\Big|p(z)\Big) - E_{q_\phi(z)}\Big[\mathcal{D}_{KL}\Big(q_\phi(x|z)\Big|\Big|p_\theta(x|z)\Big)\Big]$$

where D_{KL} is the Kullback-Leibler divergence. We include a λ prefactor which counteracts the imbalance in terms of dimensionality of the sequence space $\mathcal{X}$ and latent space $\mathcal{Z}$. For example, in our implementation, we have $\mathbf{x} \in \mathcal{R}^{59\times 21}$ and $\mathbf{z} \in \mathcal{R}^3$. To achieve an Information Maximizing VAE (InfoVAE), we will add a mutual information term $\mathcal{I}_q(x;z)$ so that the above equation becomes:

$$\mathcal{L}_{InfoVAE} = -\lambda\mathcal{D}_{KL}\Big(q_\phi(z)\Big|\Big|p(z)\Big) - E_{q_\phi(z)}\Big[\mathcal{D}_{KL}\big(q_\phi(x|z)\Big|\Big|p_\theta(x|z)\Big)\Big] + \alpha\mathcal{I}_q(x;z)$$

where $\mathcal{I}_q(x;z)$ and α encourages the model to use the latent codes, potentially avoiding posterior collapse, and weighing the influence of this mutual information term accordingly. Since the above $\mathcal{L}_{InfoVAE}$ expression cannot be directly optimized, we can rewrite it into an equivalent form which can be optimized. By using the following definitions $\mathcal{I}_q(x;z) = E_{q_\phi(x,z)}\Big[log\frac{q_\phi(x,z)}{q_\phi(x)q_\phi(z)}\Big] = -E_{q_\phi(x,z)}\Big[log\frac{q_\phi(z)}{q_\phi(z|x)}\Big]$ and the fact that $q_\phi(x|z) = p_{\mathcal{D}}(x)q_\phi(z|x)/q_\phi(z)$, we can rewrite the objective as follows:

$$\mathcal{L}_{InfoVAE} = E_{q_\phi(x,z)}\Big[-\lambda log\frac{q_\phi(z)}{p_\theta(z)} - log\frac{q_\phi(x|z)}{p_\theta(x|z)} - \alpha log\frac{q_\phi(z)}{q_\phi(z|x)}\Big]$$

$$= E_{q_\phi(x,z)}\Big[log p_\theta(x|z) - log\frac{q_\phi(z)^{\lambda+\alpha-1}p_{\mathcal{D}(x)}}{p_\theta(z)^\lambda q_\phi(z|x)^{\alpha-1}}\Big]$$

$$\begin{aligned}\mathcal{L}_{InfoVAE} = E_{P_D(x)}E_{q_\phi(z|x)}\Big[log\big(p_\theta(x|z)\big)\Big] - \big(1-\alpha\big)E_{P_D(x)}\Big[\mathcal{D}_{KL}\Big(q_\phi(z|x)\Big|\Big|p_\theta(z)\Big)\Big] - \\ (\alpha+\lambda-1)\mathcal{D}_{KL}\Big(q_\phi(z)\Big|\Big|p_\theta(z)\Big) - E_{P_D(x)}\Big[log\Big(p_D(x)\Big)\Big]\end{aligned} \tag{3.4}$$

where $E_{P_{\mathcal{D}}(x)}\Big[log\Big(p_D(x)\Big)\Big]$ is a constant with no trainable parameters that can be omitted since it does not play a role in terms of the loss gradient $\nabla\mathcal{L}_{InfoVAE}$. For our implementation, we find setting the hyperparameters $\alpha = 1$ and $\lambda = 2$ perform quite well in terms of sequence reconstruction and novel design generation. Thus, the overall expression becomes:

$$\mathcal{L} = E_{P_D(x)}E_{q_\phi(z|x)}\Big[log\Big(p_\theta(x|z)\Big)\Big] + 2\mathcal{D}_{KL}\Big(q_\phi(z)||p(z)\Big) = \mathcal{L}_{Recon} + 2\mathcal{L}_{KL} \tag{3.5}$$

Furthermore, we can swap out the KL-divergence loss with a strict divergence loss, in particular the max-mean discrepancy which quantifies the distance between two distributions by comparing all of their moments when implementing the kernel embedding trick with a characteristic kernel [Gretton et al., 2006, Li et al., 2015, Dziugaite et al., 2015]. Thus, the regularized term $\mathcal{L}_{KL}$ is replaced with the following expression:

$$\begin{aligned}\mathcal{L}_{MMD} = \mathcal{D}_{MMD}\Big(q_\phi(z)|p(z)\Big) = E_{p(z),p(z')}\big[k(z,z')\big] - \\ 2E_{q(z),p(z')}\big[k(z,z')\big] + E_{q(z),q(z')}\big[k(z,z')\big]\end{aligned} \tag{3.6}$$

where $k(\cdot,\cdot)$ is a positive definite kernel and $D_{MMD} = 0$ if and only if $p(z) = q(z)$. We choose the radial basis function (i.e., Gaussian) kernel $k(z,z') = e^{(z-z')^2/\sigma^2}$ as our characteristic kernel $k(\cdot,\cdot)$. We found that setting σ equal to the size of the latent space led to adequate performance in learning a continuous latent space, leading to excellent generative performance via sampling $\mathbf{z}$ vectors and decoding protein sequences $\mathbf{x}$.

Both the encoder and decoder are implemented as fully connected feedforward artificial neural networks with three hidden layers (En1, En2, En3 for encoder and De1, De2, De3 for decoder). Two Dropout layers are employed between (En1, En2) and (De2, De3) with dropout hyperparameters of $p = 0.3$ and $p = 0.7$. The number of units in each hidden layer is 1.5 times length of the one-hot encoded sequence ($59 \times 21 = 1239$). The activation function between linear layers along the encoder is leaky ReLU with 0.1 negative slope hyperparameter, while the activation function is simple ReLU functions between linear layers along the decoder. The final activation function for the decoder is a softmax function, which maps the logits to categorical probability distributions for each amino acid position along the whole sequence. We used Tensorflow [Abadi et al., 2016] and Keras [Chollet, 2018] to implement our MMD-InfoVAE model and trained our model using ADAM optimizer [Kingma and Ba, 2014] with a learning rate of 0.0001. We used a 3D latent space based on results of five-fold cross validation [Kohavi, 1995] by taking into consideration both validation error and gap between training and validation error. Training was conducted for 1000 epochs with batch size equal to 128 where validation loss stopped decreasing. Codes for the MMD-InfoVAE model are available at `https://github.com/Ferg-Lab/Protein_design_mmdVAE_torch` and chapter 7.

Global sampling from the trained MMD-InfoVAE model was conducted by randomly sampling 2000 latent vectors from the Gaussian prior $p(z)$, $z \sim \mathcal{N}(0, I)$. We passed each latent vector z through the trained neural network decoder $p_\theta(x|z)$ to convert these into complete protein sequence with amino acid labels. We converted and decoded probabilities along each amino acid position to amino acid labels by using the argmax function, which assigns the amino acid based on the highest probability label. Local sampling was conducted by randomly sampling 1000 latent embeddings from an anisotropic Gaussian distribution estimated by the functional Sho1 embedded orthologs in the 3D latent space. The sequences were then filtered to eliminate highly similar sequence resulting in the production of 2000 globally-sampled sequences and 987 locally-sampled sequences.

CHAPTER 4
EXPANDING THE FUNCTIONAL SPACE OF NATURAL PROTEINS

4.1 Spatial characteristics of the Sho1-SH3 function in the infoVAE latent space

The generative efficiency of the infoVAE latent space inspires a deeper study of how Sho1^{SH3} function maps to latent space position. As noted, the functional natural Sho1^{SH3} and synthetic orthologs are tightly localized to a radially extended wedge-like structure in the VAE latent space (Fig. 14). To make this quantitative, we defined a minimal polygon in the latent space (a so-called "convex hull") that bounds the natural sequences displaying full function in the *S. cerevisiae* Sho1 pathway (Fig. 15A). The majority of Sho1^{SH3} orthologs in the fungal kingdom (155/172) lie within the hull, and very few sequences within the hull are not functional (Fig. 15B). Also, synthetic orthologs embedding inside the hull show the same distribution of function as their natural counterparts (Fig. 15C-D). Thus, the hull represents a bounding box that defines the space of extant and synthetic functional Sho1^{SH3}-like orthologs.

How does Sho1^{SH3}-like function change as one exits the convex hull? Consistent with the idea that the hull defines Sho1^{SH3} function, synthetic orthologs re-embedding outside the convex hull are largely non-functional, with the few that do show Sho1^{SH3}-like function occurring in the immediate shell outside the hull (Fig. 15E-F). To quantitatively examine how Sho1^{SH3} function varies across the boundary of the hull, we computed the probability of functional sequences in the *S. cerevisiae* Sho1 pathway as a function of scaled volume shells of the convex hull moving from within the hull to outside (Fig. 15G-H). The data show that Sho1^{SH3}-like function drops sharply across the boundary, supporting the idea that the hull largely encloses the sequence rules for Sho1^{SH3} function.

An interesting feature is that the immediate environment outside the convex hull includes some bonafide Sho1$^{\mathrm{SH3}}$ synthetic orthologs (Fig. 15E, yellow symbols). This demonstrates a principle of extrapolation in the VAE model in which the space of designable functional sequences extends beyond the limits defined by natural orthologs alone.

4.2 Locality in the latent space exposes global amino acid constraints

The finding that locality within the convex hull of the InfoVAE latent space defines Sho1$^{\mathrm{SH3}}$ function provides an opportunity to examine the pattern of amino acid constraints that specifically underlie orthologous function. A simple approach is to compare the conservation of sequence positions in sequences sampled globally from the VAE latent space with that from sequences embedded within the convex hull (Fig. 16). In essence, this analysis provides as first-order view of where the "extra" constraints to be a Sho1$^{\mathrm{SH3}}$ ortholog occur in the amino acid sequence. The conservation pattern for globally sampled sequences is nearly the same as for the natural MSA (Fig. Sxx), a result consistent with the finding that global design reproduces the distribution of function in the natural MSA. However, it is quite different for sequences sampled within the convex hull bounding Sho1$^{\mathrm{SH3}}$-like function (Fig. 16A). The differences in conservation can be modeled by a double Gaussian mixture model, providing a statistical basis to identify positions that contribute the most to Sho1 function (Fig. 16B). The extra constraints for Sho1$^{\mathrm{SH3}}$ function arise both at known specificity determining sites in the ligand binding pocket Feng et al. [1994], Saksela and Permi [2012] and at a set of weakly-conserved and solvent-exposed positions distributed throughout the protein structure (Fig. 16C). These findings illustrate the use of VAE models to provide new hypotheses for mechanisms of protein function in specific cellular contexts *in vivo*.

4.3 Methods

4.3.1 Convex hull analysis

To inspect relationship between location in the latent space and functionality, convex hull analysis was performed through the scipy.spatial.ConvexHull method [Virtanen et al., 2020] with a tolerance of 10^{-12}. The hull in the latent space was defined by all functional sequence (r.e. > 0.5) in the whole dataset of 5299 sequences. For outlier removal, since these functional sequences do not form any well-defined distribution, we excluded sample points with latent coordinate $z_0 < (-0.4)$ or $z_1 > 0.0$.

4.3.2 Calculation of Kullback-Leibler relative entropy

Computation of D is based on our previous work of Statistical Coupling Analysis (SCA) [Rivoire et al., 2016]. For position i in our MSA, we have

$$D_i = \sum_{a=0}^{20} f_i^a \ln \frac{f_i^a}{\bar{q}^a} \tag{4.1}$$

where $\bar{q}^a = (1 - \bar{q}^0) q^a$, $\bar{q}^0$ represents the fraction of gaps in the alignment, and q^a is the background distribution of amino acid a computed over the non-redundant database of protein sequences. f_i^a is the observed frequency of amino acid a at position i in the MSA where length of each sequence is 59.

CHAPTER 5
EXAMINING THE PRINCIPLES OF POSITIVE AND NEGATIVE DESIGN OF ORTHOLOGS AND PARALOGS OF SH3 DOMAINS

The evolutionary constraints on proteins can involve positive design, selection for biochemical function, and negative design, selection against biochemical functions. Previous research on the Sho1$^{\mathrm{SH3}}$ domain in *S. cerevisiae* osmosensing pathway has proposed a model to understand these natural constraints. Substituting the Sho1$^{\mathrm{SH3}}$ domain with any of the 26 paralogous SH3 domains fails to rescue growth under 1M KCl conditions (although all SH3 domains exhibit equal growth under 0M KCl). However, replacing the Sho1$^{\mathrm{SH3}}$ domain with an ensemble of mammalian SH3 domains does provide some level of rescue [Zarrinpar et al., 2003]. Moreover, the growth rates were found to be proportional to the binding free energy between the SH3 variants and the Pbs2 target ligand (1C). Therefore, it is argued that the specificity of the SH3 domains is a result of both positive and negative design, and that evolutionarily distant domains from the *S. cerevisiae* genome have lost negative selection.

This work raises an interesting question about the rate of divergence of SH3 positive and negative design. Is selection for orthologous Sho1$^{\mathrm{SH3}}$ domains to bind Pbs2 more conserved than the selection for the paralogs to not bind Pbs2? And in that context, how are positive and negative design encoded in the pattern of constraints on amino acids? These questions relate to a more general issue of the rate of functional variation in orthologs versus paralogs of a protein family. Orthologs are homologs that are the result of divergence after a speciation event. In contrast, paralogs are homologs that are the result of divergence after a duplication event. Orthologs are typically proteins found in the same functional context (multidomain protein, pathway, or biological process) in many different species. For example, the SH3 domains attached to the transmembrane domains of the Sho1 receptor (Sho1$^{\mathrm{SH3}}$) in many species can operationally be defined as orthologs.

But at a biochemical level, are orthologs more functionally similar to each other than to paralogs? Orthologs are commonly thought to be more similar in function than paralogs, following the "standard model" of orthology. However, some have questioned the validity of this assertion. For example, in steroid hormone receptors, some orthologs have diverged to bind different ligands, while some paralogs have acquired the capacity to bind the same ligand. Hence, orthologs can vary, and paralogs need not necessarily vary much, challenging the notion of a fundamental difference between orthologs and paralogs.

In the case of Sho1$^{\mathrm{SH3}}$, it may be interesting to have a model system that allows clear, distinct, and measurable definitions of function in orthologs and paralogs. This would enable a systematic study of the divergence of these properties as a function of evolutionary distance. In this context, orthologs of Sho1$^{\mathrm{SH3}}$ can be defined as those SH3 domains that present in the Sho1 receptor, and paralogs as those SH3 domains attached to other molecules. The "function of orthologs" can be defined as the capacity to support growth in high-osmolar external conditions (by binding Pbs2, positive design), while the "function of paralogs" can be defined as not carrying out that function (not binding Pbs2, negative design). Investigating whether paralogs show similar or different divergence of function compared to orthologs as a function of evolutionary distance can provide insights into the implementation of various aspects of fitness, such as how fitness is positively and negatively designed in natural proteins.

5.1 Orthology and Phylogeny in VAE Latent Spaces

We firstly embedded the SH3 sequences into VAE latent spaces to identify potentially insightful evolutionary patterns. Fig. 5 and 6 shows the structure of the 3D InfoVAE latent space for the SH3 family. Interestingly, annotation shows that phylogeny is not the primary organizing principle [Ding et al., 2019]. For example, SH3 sequences from the Saccaromycotina family, the Pezizomycotina class, and the Basidiomycota division are distributed throughout the latent space with no immediately obvious pattern of localization (Fig. 5A). In contrast, sequences are more distinctly organized by paralog group in the fungal genomes. The (Bzz1_1, Abp1, Rvs167, and Sho1 SH3 domains fall into distinct wedge-like divisions of the latent space (Fig. 5B and 6). However, within each paralog wedge, a sub-organization by phylogeny is evident. For example, for the Sho1^{SH3} group, the Ascomycota and Basidomycota divisions form two branches extending radially from the origin of the latent space, and the non-dikarya SH3 domains are more proximal. The precise meaning of the spatial distribution within the patterns is a matter for further study, but we can conclude that the InfoVAE produces a hierarchical organization of SH3 homologs in which functional distinctions are primary, and phylogeny is secondary. In the mean time, the vanilla VAE latent space shows a similar hierarchical clustering (Fig. 8).

Further more, we used the trained InfoVAE encoder to embed sequences with only intrinsic constraints at the bmDCA generated sequences into the latent space. These embeddings test how sequences made with just first- and second-order MSA statistics are represented (Fig. 4). The data show that these sequences localize closer to the origin of the VAE latent space, with no observed probability density in the peripheral regions that best distinguish the fungal paralog groups (Fig. 5B and 6). Note that the VAEs are trained to produce latent space that are multi-dimensional Gaussians; thus, the basic result here is that the bmDCA sequences tend towards the average position in latent space. In contrast, VAE sequences extend to more unique positions in the tails of the distribution. These findings suggest that the VAE is learning a different and potentially deeper representation of the information content of SH3 sequences.

5.2 Relationship between Osmosensing Function and Evolutionary Distance from *S. cerevisiae*

To explore the connection between osmosensing function and the evolutionary distance of the SH3 domain from *S. cerevisiae* genome, we conducted a high-throughput yeast osmosensing assay on 7865 natural SH3 sequences. The results are depicted in Figure...

CHAPTER 6
IMPLICATIONS FOR FUTURE RESEARCH IN PROTEIN DESIGN

We investigated the diversity and specificity of the SH3 protein family and their role in protein-protein interactions and cellular signaling using the yeast osmosensor, Sho1^{SH3}, as a model. Various experimental techniques, such as high-throughput osmosensing assays and in vitro SH3 tests, were employed to acquire robust experimental data. This data was used to build data-driven models, including bmDCA and VAEs for designing novel functional osmosensing orthologs. The designed sequences reproduced the diversity, and expanded the functional space of natural proteins. In this last chapter, we will discuss the impact of the computational models and the experimental systems built in this study, and point out future directions of research based on our findings.

6.1 Advantages and Impact of the Experimental Model System

The high-throughput yeast osmosensing experimental system has proven itself as a powerful tool in the fields of protein design and understanding protein binding properties. This system allows for the straightforward design of SH3 domains based on model sequences and readily facilitates their testing. One of the system's paramount advantages is its efficiency. The entire process, encompassing gene cloning, yeast transformation accumulation across 2-3 batches, and the selection assay, takes merely a week each. This streamlines the process and ideally allows for a full iteration to be completed in just a month. Furthermore, the osmosensing system is cost-effective. The absence of expensive reagents dramatically reduces the costs associated with each iteration. The requirements are limited to an osmosensitive yeast strain and a vector containing the Sho1 gene.

This experimental system facilitates research on data-driven, iterative optimization of protein function. Data obtained from the assay can be promptly incorporated back into the model to enhance its accuracy and predictability, effectively creating a loop of continuous improvement.

The practical utility and effectiveness of this experimental workflow have been validated by the several data-driven protein design studies we have conducted using this assay [Lian et al., 2022, Fields et al., 2023, Praljak et al., 2023]. These studies underscore the system's ability to expedite research in protein design and binding properties, demonstrating its significant potential in advancing the field.

6.2 Deep Mutational Scan for Different Paralogs for the Same Function

DMS provides a quantitative measure of the functional effects of mutations. Subramanian et al. conducted a DMS experiment on the wild-type *S. cerevisiae* Sho1$^{\mathrm{SH3}}$ domain Subramanian [2017], revealing a bimodal distribution, indicating that the majority of mutations across most positions are neutral. What about the behavior for different SH3 homologs for osmo-sensing function? It would be intriguing to carry out DMS experiments for several other SH3 domains:

(1) a partial rescuing Sho1$^{\mathrm{SH3}}$ in *Basdiomycota*;

(2) a partial rescuing Hof1$^{\mathrm{SH3}}$ in *Basdiomycota.*

By analyzing the DMS outcomes of distantly related *Basidiomycota* SH3 domains, we can gain insights into their functional diversity and potentially uncover novel allosteric effects. It is possible that these domains exhibit different responses to mutations, which could contribute to variations in their overall functional outcomes.

Furthermore, we are interested in investigating whether specific mutations in these *Basidiomycota* SH3 domains could result in a gain-of-function phenotype. Understanding the potential for gain-of-function mutations is crucial as it sheds light on the evolutionary paths and adaptive strategies employed by these domains.

Finally, it would be intriguing to show which positions within these *Basidiomycota* SH3 domains contribute to the diminishing of "design effects." Identifying the specific positions that are more tolerant of mutations or have a greater impact on functional outcomes will provide valuable insights into the structural and functional constraints acting on these domains.

By conducting DMS experiments on *Basidiomycota* SH3 domains, we aim to expand our understanding of their functional effects, uncover potential allosteric mechanisms, explore gain-of-function possibilities, and identify critical positions for future design considerations. Overall, this can be a promising direction based on our work for future researchers.

6.3 Semi-supervised InfoMax VAE

In the previous chapters, we have focused on the unsupervised VAEs, which rely solely on the intrinsic structure of the protein sequence data for representation learning and generation. To expand the potential applications and capabilities of VAE guided protein design, it is worthwhile to consider the incorporation of labeled information or fitness scores (relative enrichments) within the framework. In this chapter, I will introduce the concept of semisupervised InfoVAE as a promising model for further research [Praljak and Ferguson, 2022, Praljak et al., 2023]. The Semi-supervised InfoMax Variational Autoencoder (VAE) is an advanced framework employed in protein design that leverages fitness scores of protein sequences to guide the design process. By sampling from the latent space where fitness scores are high, the approach aims to enhance the efficiency of generating functional proteins.

The loss function for the Semi-supervised InfoMax (MMD) VAE can be represented as follows:

$$\mathcal{L}_{\text{MMD}} = \mathbb{E}_{p_{\text{data}}(x)}\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{p_{\text{data}}(x)}\,\text{MMD}\left[q_\phi(z)\|p(z)\right]$$

$$\mathcal{L}_{\text{SS-MMD}} = \mathcal{L}_{\text{MMD}} + \gamma \mathbb{E}_{p_{\text{data}}(x,y)} \left[\log p_\omega(y|z)\right]$$

$\mathcal{L}_{\text{MMD}}$ represents the MMD loss term, which computes the discrepancy between the distributions of $q_\phi(z|x)$ and the prior distribution $p(z)$. The term $\mathcal{L}_{\text{SS-MMD}}$ denotes the semi-supervised MMD loss, which incorporates the MMD loss with an additional term involving labeled data. The additional term includes the likelihood of y given z in the form of $\log p_\omega(y|z)$. The parameter γ controls the importance of the supervised term (here is the fitness score) relative to the MMD loss. Sample codes of a simple SS-MMD VAE can be found at `https://github.com/Ferg-Lab/Protein_design_mmdVAE_torch` and chapter 7.

CHAPTER 7
SUPPLEMENTARY CODES

7.1 Vanilla VAE

Listing 7.1: Model Architecture for the vanilla VAE

```python
import numpy as np
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.optim as optim

class VAE(nn.Module):
    def __init__(self, q, d, n, q_n):
        super(VAE, self).__init__()
        self.hsize=int(1.5*q) # size of hidden layer
        self.q = q
        self.d = d
        self.n = n
        self.q_n = q_n

        self.en1 = nn.Linear(self.q, self.hsize)
        self.en2 = nn.Linear(self.hsize, self.hsize) #
        self.en3 = nn.Linear(self.hsize, self.hsize)
        self.en_mu = nn.Linear(self.hsize, d)
        self.en_std = nn.Linear(self.hsize, d) # Is it logvar?

        self.de1 = nn.Linear(d, self.hsize)
        self.de2 = nn.Linear(self.hsize, self.hsize) #
        self.de22 = nn.Linear(self.hsize, self.hsize)
        self.de3 = nn.Linear(self.hsize, self.q)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.tanh = nn.Tanh()
        self.softmax = nn.Softmax(dim=1)

        self.dropout1 = nn.Dropout(p=0.3)
        self.dropout2 = nn.Dropout(p=0.3)

        self.bn1 = nn.BatchNorm1d(self.hsize) # batchnorm layer
        self.bn2 = nn.BatchNorm1d(self.hsize)
        self.bn3 = nn.BatchNorm1d(self.hsize)
        self.bnfinal = nn.BatchNorm1d(self.q)

    def encode(self, x):
        """Encode a batch of samples, and return posterior parameters for each
            point."""
        x = self.tanh(self.en1(x)) # first encode
        x = self.dropout1(x)
        x = self.tanh(self.en2(x))
        x = self.bn1(x)
        x = self.tanh(self.en3(x)) # second encode
        return self.en_mu(x), self.en_std(x) # third (final) encode, return mean
            and variance

    def decode(self, z):
        """Decode a batch of latent variables"""
        z = self.tanh(self.de1(z))
        z = self.bn2(z)
        z = self.tanh(self.de2(z))
        z = self.dropout2(z)
        z = self.tanh(self.de22(z))

        # residue-based softmax
        # - activations for each residue in each position ARE constrained 0-1 and
            ARE normalized (i.e., sum_q p_q = 1)
        z = self.bn3(z)
        z = self.de3(z)
        z = self.bnfinal(z)
        z_normed = torch.FloatTensor() # empty tensor?

        for j in range(self.n):
            start = np.sum(self.q_n[:j])
            end = np.sum(self.q_n[:j+1])
            z_normed_j = self.softmax(z[:,start:end])
            z_normed = torch.cat((z_normed,z_normed_j),1)
        return z_normed

    def reparam(self, mu, logvar):
        if self.training:
```

```python
            std = logvar.mul(0.5).exp_()
            eps = std.data.new(std.size()).normal_() # torch variable
            return eps.mul(std).add_(mu)
        else:
            return mu

    def forward(self, x):
        """Takes a batch of samples, encodes them, and then decodes them again to
            compare."""
        mu, logvar = self.encode(x.view(-1, self.q)) # get mean and variance
        z = self.reparam(mu, logvar)
        return self.decode(z), mu, logvar

    def loss(self, reconstruction, x, mu, logvar):
        """ELBO assuming entries of x are binary variables, with closed form
            KLD."""
        bce = torch.nn.functional.binary_cross_entropy(reconstruction, x.view(-1,
            self.q))
        KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
        # Normalise by same number of elements as in reconstruction
        KLD /= x.view(-1, self.q).data.shape[0] * self.q
        return bce + KLD

    def get_z(self, x):
        """Encode a batch of data points, x, into their z representations."""
        mu, logvar = self.encode(x.view(-1, self.q))
        return self.reparam(mu, logvar)
```

Listing 7.2: Train the vanilla VAE

```python
import sys
import numpy as np
import argparse
import os
import time
import pickle
from model import *
import matplotlib as mpl


def VAEtrain(model, epoch, batches_per_epoch, v_train, v_val):

    # optimizer
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=0)

    # batching and training
    ind = np.arange(v_train.shape[0])
    for i in range(batches_per_epoch):
        data = torch.FloatTensor(v_train[np.random.choice(ind, size=batch_size)])
            # randomly sample training set
        data = data.to(device)
        optimizer.zero_grad()
        pred, mu, logvar = model(data)
        loss = model.loss(pred, data, mu, logvar) #loss(self, reconstruction, x,
            mu, logvar)
        loss.backward()
        optimizer.step() # optimize function...

    # training loss
    data = torch.FloatTensor(v_train)
    data = data.to(device)
    pred, mu, logvar = model(data)
    train_loss = model.loss(pred, data, mu, logvar)
    train_loss = train_loss.cpu().detach().numpy() # network is trained on this
        loss, maximize P(X), what the network see

    diff = pred.cpu().detach().numpy() - v_train
    train_loss_MSE = np.mean(diff**2) # mean square error per position, for human
        to see. Network does not know it.

    # validation loss
    data = torch.FloatTensor(v_val)
    data = data.to(device)
    pred, mu, logvar = model(data)
    val_loss = model.loss(pred, data, mu, logvar)
    val_loss = val_loss.cpu().detach().numpy()

    diff = pred.cpu().detach().numpy() - v_val
    val_loss_MSE = np.mean(diff**2)

    if (epoch % 10 == 0):
       print('====> Epoch %d done! Train loss = %.2e, Val loss = %.2e, Train loss
           MSE = %.2e, Val loss MSE = %.2e' %
           (epoch,train_loss,val_loss,train_loss_MSE,val_loss_MSE))

    return train_loss, val_loss, train_loss_MSE, val_loss_MSE


def VAEtest(model, v_test):

    data = torch.FloatTensor(v_test)
    data = data.to(device)
    pred, mu, logvar = model(data) # model is VAE?

    # ELBO test loss
    test_loss = model.loss(pred, data, mu, logvar)
    test_loss = test_loss.cpu().detach().numpy()

    # MSE test loss
    diff = pred.cpu().detach().numpy() - v_test
    test_loss_MSE = np.mean(diff**2)

    return pred, test_loss, test_loss_MSE


if __name__ =='__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-n", dest ="name", default='protein', type=str,
       help="Name of your protein.")
    parser.add_argument("-e", dest ="nbepoch", default=55, type=int, help="number
       of training epochs.")
    options = parser.parse_args()
```

```

if os.environ.get('DISPLAY','') == '':
    print('no display found. Using non-interactive Agg backend')
    mpl.use('Agg')

if torch.cuda.is_available():
    print("=> Using GPU")
    print("CUDA device count =")
    print (torch.cuda.device_count())
    print("Selecting decvice = cuda:0")
    device = torch.device("cuda:0")
    print("Device name = ")
    print (torch.cuda.get_device_name(0))
else:
    print("=> Using CPU")
    device = torch.device("cpu")

# fix random seed for reproducibility
randstate = 200
np.random.seed(randstate)
torch.manual_seed(randstate)
if device == torch.device("cuda:0"):
    randstate = 2000 # RCC fela #1234
    np.random.seed(randstate)
    torch.manual_seed(randstate)
    torch.cuda.manual_seed_all(randstate)


path = '../Outputs/'
parameters = pickle.load(open(path + options.name + ".db", 'rb'))
q_n = parameters['q_n']
v_traj_onehot = parameters['onehot']

print(v_traj_onehot.shape)
print('number of possible amino acids in each position q_n = \n',q_n)
print('length(q_n) = ',len(q_n))

N=np.size(v_traj_onehot,axis=0)
q=np.size(v_traj_onehot,axis=1)
n=np.size(q_n)
idx = np.arange(N)

```

```
    test_frac = 0.01
    val_frac = 0.20
    v_train_val, v_test, idx_train_val, idx_test = train_test_split(v_traj_onehot,
        idx, test_size=test_frac, random_state=randstate)

    print ("N = %d" % N)

    v_train, v_val, idx_train, idx_val, = train_test_split(v_train_val,
        idx_train_val, test_size=val_frac/(1-test_frac), random_state=randstate)
    print ("Training starts...")

    # training final VAE over all train_val data at optimal d
    # manually modify after finding optimal training length.
    start=time.time()

    d=3
    batch_size = 40
    over_batch = 5
    batches_per_epoch =
        np.int32(over_batch*np.ceil(v_train_val.shape[0]/batch_size))
    nb_epoch = options.nbepoch # Optimal nb_epoch is 55.

    model = VAE(q, d, n, q_n).to(device)

    loss_train = []
    loss_train_MSE = []
    for epoch in range(1, nb_epoch+1):
        train_loss, _, train_loss_MSE, _ = VAEtrain(model, epoch,
            batches_per_epoch, v_train_val, v_val)
        # training together with validation set are used together to train the
            final VAE.
        loss_train.append(train_loss)
        loss_train_MSE.append(train_loss_MSE)

    end = time.time()
    print("Using device = %s" % device)
    print("Elapsed time %.2f (s)" % (end - start))

    # saving trained model
    save_path = "./VAE_"+options.name+".pyt"
    torch.save(model.state_dict(), save_path)
```

Listing 7.3: Generate novel protein sequences by the vanilla VAE

```python
#!/usr/bin/env python
# coding: utf-8

import sys
import numpy as np
import os
import argparse
import pickle
import multiprocessing as mp
import time
import shutil
from itertools import repeat

import toolkit
from model import *

from Bio import SeqIO
from Bio import AlignIO


if __name__ =='__main__':
    parser = argparse.ArgumentParser(description='Hint: In total ngen*nsamp new
        sequences are generated, default 1000. Then they are filtered according to
        thresholds of minimum Hamming distance.')

    parser.add_argument("-g", "--ngen", dest ="ngen",
                        default=1000, type=int,
                        help="times of sampling in the latent space. Default 1000.
                            Recommended to enter a multiple of 10.")
    parser.add_argument("-s", "--nsamp", dest ="nsamp",
                        default=10, type=int,
                        help="times of throwing dice at each sampling point. Default
                            10")
    parser.add_argument("-r", "--randseed", dest ="randseed",
                        default=1000, type=int, help="Random seed. Default 1000.")
    parser.add_argument("-n", "--name", dest ="name",
                        default='protein', type=str, help="Name of your protein.")
    parser.add_argument("-c", "--custom", dest ="custom",
                        default='', type=str,
                        help="A custom string for your generated sequence file name.
```

```
                         Default None.")

    parser.add_argument("-a", "--sca", dest="sca", action="store_true",
                        default=False, help="Compute SCA for generated sequecnes")

    options = parser.parse_args()
    device = torch.device("cpu")
    torch.manual_seed(20)

    print('Loading data...')
    path = '../Outputs/'
    parameters = pickle.load(open(path + options.name + ".db", 'rb'))
    q_n = parameters['q_n']
    aaindex = parameters['index']
    v_traj_onehot = parameters['onehot']
    records_MSA = parameters['seq']

    N=np.size(v_traj_onehot,axis=0)
    q=np.size(v_traj_onehot,axis=1)
    n=np.size(q_n)

    print('Loading VAE...')
    d=3
    model = VAE(q, d, n, q_n)
    model.load_state_dict(torch.load('VAE_SH3.pyt',map_location='cpu'))
    model.eval()

    # Generate new sequences
    start_all = time.time()

    seed = options.randseed
    n_gen = options.ngen
    n_sample = options.nsamp

    np.random.seed(seed)
    real_nohot_list = toolkit.convert_nohot(v_traj_onehot, q_n)
    seed_list = np.random.randint(0, 2**32, 10)
    #pool = mp.Pool(mp.cpu_count())

    print('Start generating sequences...')
    st_time = time.time()

    np.random.seed(seed)
    z_gen = np.random.normal(0., 1., (n_gen, d)) #generate normal distribution of random numbers
    data = torch.FloatTensor(z_gen).to(device)
    data = model.decode(data) # Use the decoding layer to generate new sequences.
    v_gen = data.cpu().detach().numpy()
    sample_list = []
    z_list = []

    for i in range(int(n_gen/10)):
        for k in range(n_sample):
            v_samp_nothot = toolkit.sample_seq(seed+k, q, n, q_n, i, v_gen)
            sample_list.append(v_samp_nothot)
            z_list.append(z_gen[i])

    alp_new_seq = toolkit.convert_alphabet(np.array(sample_list), aaindex, q_n)
    end_time = time.time()
    print("Elapsed time %.2f (s)" % (end_time - st_time))

    print(’Computing VAE logP for selected sequences...’)
    st_time = time.time()

    print(’Converting generated sequences to Potts...’)
    new_potts, _ = toolkit.convert_potts(alp_new_seq, aaindex)
    print(’Reconstructing with VAE...’)
    pred_ref,_,_ = model(torch.FloatTensor(new_potts))
    p_weight = pred_ref.cpu().detach().numpy()
    print(’computing logP...’)
    log_norm = toolkit.make_logP(new_potts, p_weight, q_n)

    if options.sca:
        print(’Start computing SCA...’)
        filename = options.name+ options.custom+’_sca’
        if os.path.isdir(’output’)==0:
            os.mkdir(’output’)
        with open(’output/’ + filename+’.fasta’, ’w’) as f:

            # write the reference sequence
            f.write(">2vkn_chainA_p001\n")
            f.write("NFIYKAKALYPYDADDAYEISFEQNEILQVSDIEGRWWKARRNGETGIIPSNYVQLIDG\n") #2vkn_chainA_p001

            for item in alp_new_seq[:-1]:
                f.write(">gi\n")
                f.write("%s\n" % item)
        os.system('scaProcessMSA -a output/' + filename +'.fasta -b data -s 2VKN
            -c A -p 0.3 0.2 0.2 0.8')
        # Note: the above line should be customes based on the protein family you
            chose
        os.system('scaCore -i output/' + filename +'.db')
        os.system('scaSectorID -i output/' + filename +'.db')

        if os.path.isfile(path + filename +'.db'):
            os.remove(path + filename +'.db')
        os.rename('output/'+filename +'.db',path + filename +'.db')
        shutil.rmtree('output')
        print('SCA computing finished.')

    end_time = time.time()
    print("Elapsed time %.2f (s)" % (end_time - st_time))

    np.savez(path + options.name + options.custom + 'gen_data.npz', seq =
        alp_new_seq, ham = 0, logP = log_norm, z_list = z_list)

    end_all = time.time()
    print("\nTotal elapsed time %.2f (s)" % (end_all - start_all))
```

7.2 InfoMax VAE

Listing 7.4: Model Architecture for Unsupervised and Supervised InfoVAEs

```python
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
from sklearn.model_selection import train_test_split

class Encoder(nn.Module):

    def __init__(self, seq_len, aa_var, zdim, alpha):
        super(Encoder, self).__init__()

        self.zdim = zdim
        self.seq_len = seq_len
        self.aa_var = aa_var
        self.alpha = alpha
        self.q = seq_len * aa_var
        self.hsize=int(1.5*self.q)
        #self.en_mu = nn.Linear(self.hsize, d)
        #self.en_std = nn.Linear(self.hsize, d)

        self.model = nn.Sequential(
            #encoder layer 1
            nn.Linear(self.q, self.hsize),
            nn.LeakyReLU(self.alpha, inplace=True),
            nn.Dropout(p=0.3),

            #encoder layer 2
            nn.Linear(self.hsize, self.hsize),
            nn.LeakyReLU(self.alpha, inplace=True),
            nn.BatchNorm1d(self.hsize), # BN1

            #encoder layer 3
            nn.Linear(self.hsize, self.hsize),
            nn.LeakyReLU(self.alpha, inplace=True),

            nn.Linear(self.hsize, self.zdim)
        )
    def forward(self, x):
        x = x.view(x.size(0), self.q)
        return self.model(x)

class Decoder(nn.Module):

    def __init__(self, seq_len, aa_var, zdim, alpha):
        super(Decoder, self).__init__()

        self.seq_len = seq_len
        self.aa_var = aa_var
        self.alpha = alpha
        self.q = seq_len * aa_var
        self.zdim = zdim
        self.hsize=int(1.5*self.q)

        self.model = nn.Sequential(
            #decoder layer 1
            nn.Linear(self.zdim, self.hsize),
            nn.LeakyReLU(self.alpha, inplace=True),
            nn.BatchNorm1d(self.hsize), #BN2

            #decoder layer 2
            nn.Linear(self.hsize, self.hsize),
            nn.LeakyReLU(self.alpha, inplace=True),
            nn.Dropout(p=0.3),

            #decoder layer 3
            nn.Linear(self.hsize, self.hsize),
            nn.LeakyReLU(self.alpha, inplace=True),
            nn.BatchNorm1d(self.hsize),

            nn.Linear(self.hsize, self.q),
            #nn.BatchNorm1d(self.q), #BNfinal
        )
    def forward(self, z):
        outputs = self.model(z)
        outputs = outputs.view(z.size(0), self.seq_len, self.aa_var)
        outputs = nn.Softmax(dim = 2)(outputs)
        return outputs

class Regression(nn.Module):
    def __init__(self, zdims, omega = 10, p = 0.2):
        super(Regression, self).__init__()
        self.zdims = zdims
        self.omega = omega
        self.tanh = nn.Tanh()
        self.relu = nn.ReLU()
        self.p = p

        self.dropout = nn.Dropout(p)

        self.regressor = nn.Linear(self.zdims, self.omega)
        nn.init.xavier_normal_(self.regressor.weight)

        self.regressor_out = nn.Linear(self.omega, self.omega)
        nn.init.xavier_normal_(self.regressor_out.weight)

        self.dense_out_R = nn.Linear(self.omega, 1)
        nn.init.xavier_normal_(self.dense_out_R.weight)

    def forward(self, z):
        h_R = self.dropout(self.tanh(self.regressor(z)))
        h_R = self.dropout(self.relu(self.regressor_out(h_R)))
        out = self.dense_out_R(h_R)
        return out


def loss_function(recon_x, x, z, device_name):
    batch_size = x.size(0)
    zdim = z.size(1)
    true_samples = torch.randn(batch_size, zdim, requires_grad =
        False).to(device_name)

    loss_MMD = compute_mmd(true_samples, z)
    loss_REC = (recon_x - x).pow(2).mean()

    return loss_REC + 2*loss_MMD, loss_REC, loss_MMD

def loss_ss(recon_x, x, z, y, y_pred, device_name):
    batch_size = x.size(0)
    zdim = z.size(1)

    mask = ~torch.isnan(y)

    true_samples = torch.randn(batch_size, zdim, requires_grad =
       False).to(device_name)

    loss_MMD = compute_mmd(true_samples, z)
    loss_REC = (recon_x - x).pow(2).mean()
    loss_pred= (y[mask] - y_pred[mask]).pow(2).mean()

    return loss_REC + 2*loss_MMD + 0.5 * loss_pred, loss_REC, loss_MMD, loss_pred

def compute_kernel(x, y):
    x_size = x.size(0)
    y_size = y.size(0)
    dim = x.size(1)
    x = x.unsqueeze(1)
    y = y.unsqueeze(0)

    tiled_x = x.expand(x_size, y_size, dim)
    tiled_y = y.expand(x_size, y_size, dim)
    kernel_input = (tiled_x - tiled_y).pow(2).mean(2)/float(dim)
    return torch.exp(-kernel_input)

def compute_mmd(x, y):
    x_kernel = compute_kernel(x, x)
    y_kernel = compute_kernel(y, y)
    xy_kernel = compute_kernel(x, y)
    mmd = x_kernel.mean() + y_kernel.mean() - 2*xy_kernel.mean()
    return mmd

class MMD_VAE(nn.Module):
    def __init__(self, zdims, seq_len, aa_var, alpha):
        super(MMD_VAE, self).__init__()
        self.zdims = zdims
        self.seq_len = seq_len
        self.aa_var = aa_var
        self.alpha = alpha
        self.encoder = Encoder(self.seq_len, self.aa_var, self.zdims, self.alpha)
        self.decoder = Decoder(self.seq_len, self.aa_var, self.zdims, self.alpha)

    def forward(self, x):
        z = self.encoder(x)
        recon_x = self.decoder(z)
        return z, recon_x


class SS_MMD(nn.Module):
    def __init__(self, zdims, seq_len, aa_var, alpha):
        super(SS_MMD, self).__init__()
        self.zdims = zdims
        self.seq_len = seq_len
        self.aa_var = aa_var
        self.alpha = alpha

        self.encoder = Encoder(self.seq_len, self.aa_var, self.zdims, self.alpha)
        self.decoder = Decoder(self.seq_len, self.aa_var, self.zdims, self.alpha)

        self.regressor = Regression(self.zdims)

    def forward(self, x):
        x = x.view(x.size(0), self.seq_len*self.aa_var)
        z = self.encoder(x)

        recon_x = self.decoder(z)
        pred_y = self.regressor(z)

        return z, recon_x, pred_y
```

7.3 Others

Listing 7.5: Codes for reverse translation and verification

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


from __future__ import division
import sys
import numpy as np
import csv
import pandas as pd
from Bio.Seq import Seq
from Bio import SeqIO
import Bio.SeqUtils.MeltingTemp as mt
import scipy.io as sio
import matplotlib.pyplot as plt


# In[2]:


# function to get sequences
def get_seq(filename, get_header = False):
    records = list(SeqIO.parse(filename, "fasta"))
    records_seq = [i.seq for i in records]
    headers = [i.description for i in records]
    if get_header == True:
        return records_seq, headers
    else:
        return records_seq


# ## Get fasta file for protein sequences to be reverse-translated
#
# * Gaps in the fasta file should be removed in advance. Use *remove_gap_fasta.py*
    to remove gaps:
#
# python remove_gap_fasta.py *input_file.fasta* *output_name*
```

```
#
# example:
#
# python remove_gap_fasta.py *Inputs/test.fasta* *test2*

# In[3]:


filename = 'Final_New_Proteins_nogap.fasta'
seq, head = get_seq('Inputs/'+filename, get_header = True)
N = len(seq)

fill_to = 250


# ---
# ## Process BLAST results
# If there is already *localkeep_New_Proteins.fasta* and *lib_local.mat*, **skip these two steps.**
# 1. In this repository folder, run following commands to blast the local library:
#
# cd Inputs
# tblastn -query Final_New_Proteins_nogap.fasta -subject ../Utility/twist_red_seqs_forblast.an -max_target_seqs 1 -evalue 1e-3 -word_size 6 -outfmt 6 > blast_local.txt
#
# 2. Running following codes to remove sequences without blast results or duplicated in blast results.

# In[4]:


blast = pd.read_csv("Inputs/blast_local.txt", sep = '\t', header=None)
Nb = len(blast)
print('%d sequences blasted' %Nb)


# In[5]:


keep = []
blast_head = np.array(blast[0])
```

```
for i in range(N):
    if head[i] in blast_head:
        keep.append(i)
print('%d Sequences out of %d are kept.' %(len(keep), len(seq)))


# In[6]:


blast_head = np.array(blast[0])
for i in range(N):
    if head[i] not in blast_head:
        a = int(head[i][4:])
        #print(a)
        print(seq[a])


# In[7]:


blast_new = blast.copy()
blast_new2 = blast_new.loc[~blast_new[0].duplicated(keep='first')] # remove
    dulicated blast results

# Check header match
for i in range(len(keep)):
    assert np.array(blast_new2[0])[i] == head[keep[i]]


# In[7]:


# Write non-redundant protein list for kept after blast
with open('Outputs/localkeep_New_Proteins.fasta', 'w') as f:
    for i in range(N):
        if i in keep:
            f.write(">%s\n" %head[i])
            f.write("%s\n" %seq[i])


# In[8]:

```

```

tfile = open('Outputs/local_blast_keep.txt', 'w')
tfile.write(pd.DataFrame.to_csv(blast_new2, sep = '\t', index = 0, header = False))
tfile.close()


# In[9]:


seqg, headg = get_seq('Utility/twist_red_seqs_forblast.an', get_header = True)
Ng = len(seqg)

local_gene = []
bind_array = np.array(blast_new[1]).astype(str)
for i in range(Nb):
    for j in range(Ng):
        if bind_array[i] == headg[j]:
            local_gene.append(seqg[j])
            break


# In[10]:


a = [str(i).lower() for i in local_gene]
sio.savemat('Outputs/local_gene.mat', {'gene':a})
# genes for the local blasted result, use the matlab file to trim...


# ---
# 3. Trim genes and get alignment using the matlab codes (*local.m*).
# 4. Run following codes to remove RE sites and add assembly primers.

# In[8]:


def sampling(aa, transdict, randstate):
    # Sample the codon for a single amino acid position (return nothing if the
       position is gap)
    if aa == '-':
        return ''
    elif np.size(TransDict[aa]['frequency']) ==1:
        #For AA with only one codon
        return(TransDict[aa]['codon'])
    else:
        # for AA with more than one codons
        np.random.seed(randstate)
        sample_tmp = np.random.multinomial(1,np.array(TransDict[aa]['frequency']))
        sample_index = np.where(sample_tmp!=0)[0]
        return TransDict[aa]['codon'][sample_index][0]


# In[9]:


seqkeep, headkeep = get_seq('Outputs/localkeep_New_Proteins.fasta', get_header =
    True)
# Get genes trimmed by matlab
lib = sio.loadmat('Outputs/lib_local.mat')['lib']
CodonUsageTable = pd.ExcelFile('Utility/yeast_codon.xlsx').parse().set_index('aa')


# In[10]:


TransDict = {} # dictionary to map amino acid with codons. One AA corresponds to
    >=1 codons
for i in set(CodonUsageTable.index): # Add dictionary for each amino acid
    TransDict.update({i:CodonUsageTable.loc[i]})


# In[11]:


gene = [lib[i][0][5][0].upper() for i in range(len(seqkeep))] # top-hitting
    natural genes of the designed seqs
Nm = len(gene)


# In[12]:


# split gene by codons
gene_split = []
```

```python
for i in gene:
    assert len(i)%3 == 0, 'length error'
    length = int(len(i)/3)
    codon_split = []
    for j in range(length):
        codon_split.append(i[j*3:j*3+3])
    assert i == ''.join(codon_split)
    gene_split.append(codon_split)


# In[13]:


# Proteins to be reverse translated, use the one aligned by matlab to keep
    consistent for translation.
protein, ref = [], []
for i in range(Nm):
    protein.append(lib[i][0][4][0])
    ref.append(lib[i][0][4][2])
    assert protein[i].replace('-','') == str(seqkeep[i]).replace('-','')


# In[14]:


fw_assembly_p = 'CCGGTTGTACCTATCGAGTG'+'GGATCC' # bamH1 + forward primer
rv_assembly_p = 'GAATTC'+'GTACCTCTCCTTGCATGGTC' # EcoR1 + reverse component of
    reverse primer

resites={'GGATCC','GAATTC','AAAAA','GGGGG','CCCCC','TTTTT'}; # BamH1, EcoR1 and
    replicating pattern


# In[29]:


gene_design, problematic_index = [], []
global_rand = 1
for i in range(Nm):
    gene_tmp, flag, gene_ind, pro_ind = '', 0, 0, 0
    # Initially flag=0, if sampled gene is problematic flag=1, if it's good flag=2
    avoid_homology_para = 0
    for j in range(len(protein[i])):
        if protein[i][j] == '-':
            pass
        elif protein[i][j] != ref[i][j] or avoid_homology_para == 5:
            # Sample a new codon if AA don't match at the position.
            # protein[i]: i th designed protein
            # ref[i]: top hit natural protein of i th designed protein, according to blast result
            gene_tmp += sampling(protein[i][j], TransDict, global_rand)
            global_rand += 1
            # Use a global random number to get different sampled codon every time.
            # here the randomseed=0 problem should be solved.
            avoid_homology_para = 0
            # avoid_homology_para is a parameter to avoid homology caused by multiple designed seqs are
            # "mutated" from one natural allele. after each 5 continuous codons same with the natural allele,
            # it makes the 6th to be a newly sampled codon.
        else:
            gene_tmp += gene_split[i][gene_ind] # use the original codon if match
            avoid_homology_para += 1
        if ref[i][j] != '-': # skip gaps
            gene_ind += 1
        if protein[i][j] != '-':
            pro_ind += 1
        gene_tmp = gene_tmp.replace('U','T') # replace codon to gene

    # Special treatment to terminals of the gene to avoid unwanted RE sites at terminals.
    # This check is specific for BamHI and EcoRI. Should be modified for other RE sites.

    if protein[i].replace('-','')[0] == 'T':
        gene_tmp = 'ACA' + gene_tmp[3:]
    if protein[i].replace('-','')[0] == 'P':
        gene_tmp = 'CCG' + gene_tmp[3:]
    if protein[i].replace('-','')[-1] == 'S':
        gene_tmp = gene_tmp[:-3]+'AGC'
    if protein[i].replace('-','')[-1] == 'G':
        gene_tmp = gene_tmp[:-3]+'GGC'

    flag = 2
    for k in resites:
        if k in gene_tmp:
            flag, modify_pos = 1, int(gene_tmp.find(k)/3)
            break

    randseed = 1
    while flag ==1: # Correct problematic sequences by resampling
        gene_fix, pro_ind = '', 0
        left, right = int(modify_pos!=0), int(modify_pos<len(gene_tmp)-1) # if it's a terminal position
        for j in range(len(protein[i])):
            if pro_ind in np.arange(modify_pos-left,modify_pos+right+1):
                gene_fix += sampling(protein[i][j], TransDict, randseed+global_rand) # resampling
                #print(i,j,randseed)
            if protein[i][j] != '-':
                pro_ind += 1
        gene_tmp = gene_tmp[:(modify_pos-left)*3] + gene_fix + gene_tmp[(modify_pos+right+1)*3:]
        gene_tmp = gene_tmp.replace('U','T')

        # Redo the treatment for terminal positions...
        if protein[i].replace('-','')[0] == 'T':
            gene_tmp = 'ACA' + gene_tmp[3:]
        if protein[i].replace('-','')[0] == 'P':
            gene_tmp = 'CCG' + gene_tmp[3:]
        if protein[i].replace('-','')[-1] == 'S':
            gene_tmp = gene_tmp[:-3]+'AGC'
        if protein[i].replace('-','')[-1] == 'G':
            gene_tmp = gene_tmp[:-3]+'GGC'

        flag = 2
        for k in resites: # Check again
            if k in gene_tmp:
                flag, modify_pos = 1, int(gene_tmp.find(k)/3)
                randseed +=1
                global_rand+=1
                break
        if randseed > 80:
            flag = 2
            for k in resites:
                if k in gene_tmp:
                    problematic_index.append(i)
                    break
    gene_design.append(gene_tmp)
    if i%1000 == 0 and i!=0:
        print('%d finished...' %i)
print('Finished!')


# In[30]:


print('There are %d problematic sequences.' %len(problematic_index))


# In[31]:


# Append the RE site + primer sequences to the genes
gene_design_fill = []
for i in gene_design:
    gene_design_fill.append(fw_assembly_p + i + rv_assembly_p)


# 3. Generated random pudding sequences and append to the beginning to fill the
    gene to 300mers.
#
# * Or 250mers if all of your SH3 genes are <= (250 - 52)/3 = 66 amino acids. View
    Twist Price policy.

# In[15]:


# length of random sequences
def gen_randseq(gene):
    len_randseq = []
    for i in gene:
        len_randseq.append(fill_to - len(i))

    print('Generating random sequences...')
    randseed = 0
    randseq_list_fill = []
    for i in len_randseq:
        flag = 0
        while flag == 0:
            np.random.seed(randseed)
            randseq = ''.join(['ACTG'[j] for j in np.random.randint(0,4,i)])

            flag = 1
            for k in resites:
                if k in randseq or k[:2] in randseq[-5:]:
                    flag = 0
                    randseed +=1
                    break
        randseq_list_fill.append(randseq)
        randseed +=1
    return randseq_list_fill

def append_rand_seq(randlist, gene):
    gene_fill_final = []
    print('Appending random sequences...')
    for i in range(len(gene)):
        gene_fill_final.append(randlist[i]+gene[i])
    return gene_fill_final


# In[33]:


randseq_list_fill = gen_randseq(gene_design_fill)
gene_fill_final = append_rand_seq(randseq_list_fill, gene_design_fill)
print('Finished!')


# 4. Append null alleles

# In[34]:


nulls = ['CCGGTTGTACCTATCGAGTGGGATCCTAGATAATTTCGGCGTGGGTATGGTGGCAGGCCCCGTGGCCGGGGGA
CTGTTGGGCGCCATCTCCTTGCATGCACCATTCCTTGCGGCGGCGGTGCTCAACGGCCTCAACCTACTACTGGGCTGCTTCCT
AATGCAGGAGTCGCATAAGGGAGAGCGTCGAGATGAATTCGTACCTCTCCTTGCATGGTC',
'CCGGTTGTACCTATCGAGTGGGATCCTAGTTAATTTCGGCGTGGGTATGGTGGCAGGCCCCGTGGCCGGGGGACTGTTGGGC
GCCATCTCCTTGCATGCACCATTCCTTGCGGCGGCGGTGCTCAACGGCCTCAACCTACTACTGGGCTGCTTCCTAATGCAGGA
GTCGCATAAGGGAGAGCGTCGAGATGAATTCGTACCTCTCCTTGCATGGTC',
'CCGGTTGTACCTATCGAGTGGGATCCTAGCTAATTTCGGCGTGGGTATGGTGGCAGGCCCCGTGGCCGGGGGACTGTTGGGC
GCCATCTCCTTGCATGCACCATTCCTTGCGGCGGCGGTGCTCAACGGCCTCAACCTACTACTGGGCTGCTTCCTAATGCAGGA
GTCGCATAAGGGAGAGCGTCGAGATGAATTCGTACCTCTCCTTGCATGGTC']

rand_for_null = gen_randseq(nulls)

nulls_fill = [rand_for_null[i]+nulls[i] for i in range(3)]


# 5. Write down the
#   * Final list of designed proteins in the library
#   * Final list of the filled oligos (250 or 300mer)

# In[41]:


with open('Outputs/Final_New_Proteins_tosubmit.fasta', 'w') as f:
    for i,item in enumerate(seqkeep):
        f.write(">%s\n" %headkeep[i])
        f.write("%s\n" %seqkeep[i])


# In[44]:


with open('Outputs/oligo_fill.an', 'w') as f:
    for i,item in enumerate(gene_fill_final):
        f.write(">%s\n" %headkeep[i])
        f.write("%s\n" %item)
    for i in range(3):
        f.write(">null%d\n" %i)
        f.write("%s\n" %nulls_fill[i])


# In[26]:


print('Writing excel file...')
oligo = pd.DataFrame(data={'header': headkeep,'gene':gene_fill_final})
oligo.to_excel('Outputs/oligo_fill.xlsx')
```

```python
# ## Check the final oligo
#
# Make sure of everything!
# 1. Check oligo structure


# In[16]:


tmp = pd.read_excel('Outputs/oligo_fill_addNULL.xlsx').iloc[:-3,:]
tmp_protein = get_seq('Outputs/Final_New_Proteins_tosubmit.fasta')
oligolist = tmp.gene


# In[29]:


frag_digest = []
for i in range(len(oligolist)):
    oligo = oligolist[i]
    frag_digest.append  (oligo[oligo.find('CCGGTTGTACCTATCGAGTGGGATCC')+26 :
        oligo.find('GAATTCGTACCTCTCCTTGCATGGTC')])


# In[18]:


# oligo length after digestion. check if (length % 3 == 0)

a = [len(i) for i in frag_digest]
plt.hist(a,30,edgecolor='k')
plt.xlabel('Length')
plt.ylabel('Count')
plt.show()


# In[19]:


for i in range(len(oligolist)):
    assert len(oligolist[i]) == fill_to

    # check if there is no RE sites, primers, replicated AAs in the oligos
    for j in ['GGATCC', 'GAATTC', 'CCGGTTGTACCTATCGAGTGG', 'GTACCTCTCCTTGCATGGTC']:
        assert oligolist[i].count(j)==1,i
    for j in
        ['GGGGG','AAAAA','TTTTT','CCCCC','GACCATGCAAGGAGAGGTAC','CCACTCGATAGGTACAACCGG']:
        assert j not in oligolist[i]

    oligo = oligolist[i]
    frag_digest = oligo[oligo.find('CCGGTTGTACCTATCGAGTGGGATCC')+26 :
        oligo.find('GAATTCGTACCTCTCCTTGCATGGTC')]
    translate = Seq(frag_digest).translate()

    assert len(frag_digest)%3 ==0
    assert str(translate) == str(tmp_protein[i])#.replace('-','') # translated
        match protein


# In[73]:


for num, i in enumerate(oligolist):
    for j in
        ['CGGTTGTACCTATCGAGT','ACCATGCAAGGAGAGGTA','TACCTCTCCTTGCATGGT','ACTCGATAGGTACAACCG']:
        for k in range(1,17):
            for h in ['A','C','T','G','AA','AC','AT','AG','CA','CC','CT','CG',
                      'TA','TC','TT','TG','GA','GC','GT','GG']:
                check_length = len(randseq_list_fill[num]) + 18
                assert j[:k]+h+j[k+1:] not in str(i)[:check_length],i


# 2. GC ratio

# In[20]:


gc_ratio = []
for i in oligolist:
    r=(i.count('G')+i.count('C'))/fill_to
    gc_ratio.append(r)
    if r==0:
        print(i)
        break
```

```
plt.hist(gc_ratio,20,edgecolor='k')
print('Max GC ratio = %.2f; Min GC ratio = %.2f' % (max(gc_ratio),min(gc_ratio)) )
plt.title('library 1')
plt.xlabel('GC ratio')
plt.ylabel('Count')
plt.show()


# 3. Codon usage

# In[21]:


frag_digest=[]
gene_split_new = []
for num, i in enumerate(oligolist):
    frag_digest.append(i[i.find('CCGGTTGTACCTATCGAGTGGGATCC')+26 :
        i.find('GAATTCGTACCTCTCCTTGCATGGTC')])
    length = int(len(frag_digest[num])/3)
    for j in range(length):
        gene_split_new.append(frag_digest[num][j*3:j*3+3])


# In[22]:


plt.figure(figsize=[28,5])
plt.hist(gene_split_new,bins=122)
plt.title('New')
plt.show()


# In[23]:


gene_split_nat = [item for sublist in gene_split for item in sublist]
plt.figure(figsize=[28,5])
plt.hist(gene_split_nat,bins=122)
plt.title('Natural')
plt.show()
```

Listing 7.6: local.m used for trimming genes and getting alignment

```
lib=fastaread('Outputs/localkeep_New_Proteins.fasta');
localgene = load('Outputs/local_gene.mat').gene;
hits=blastreadlocal('Inputs/blast_local.txt', 8);

for i=1:numel(lib)
seq=lib(i).Sequence;
seq(seq=='-')=[];
lib(i).protein=seq;
lib(i).len=numel(seq);
end
% now, retrieve the sequences from ncbi % load('rid.mat');

keep=zeros(size(lib));
for i=1:numel(hits)
    percent=hits(i).Hits(1).HSPs(1).Identities(1).Percent;
    if percent==100
        keep(i)=1;
    end
    [mper, ind]=max(percent);
    if mper ==100
        hits(i).Hits(1:ind-1)=[]; keep(i)=1;
        disp([num2str(i) ' Corrected']);
    else
        disp([num2str(i) ' Fail']);
    end
end

per=zeros(1,1);
j=1;
per(j)=hits(i).Hits(j).HSPs(1).Identities(1).Percent;
    for i=1:numel(lib)
        region=hits(i).Hits(1).HSPs(1).SubjectIndices;
        if region(2) < region(1) % reverse frame
            region=sort(region);
            sequence{i}=seqrcomplement(localgene(i,region(1):region(2)));
        else
            sequence{i}=localgene(i,region(1):region(2));
        end

        prot{i}=nt2aa(sequence{i},'AlternativeStartCodons','False','ACGTOnly','False');
        [a1,a2,a3]=nwalign(lib(i).Sequence, prot{i});
        lib(i).a2=a2;
        match(i)=sum(a2(2,:)=='|')-lib(i).len;
        disp(i);
    end
% turns out, some 16 hits are not good matches. i'll remove them from lib.

for i=1:numel(lib)
    lib(i).dna=sequence{i};
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%save('lib_local.mat','lib')

lib1=lib;
save('Outputs/lib_local.mat','lib')
```

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

Ethan C Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16(12):1315–1322, 2019.

D Altschuh, T Vernet, P Berti, D Moras, and K Nagai. Coordinated amino acid changes in homologous protein families. *Protein Engineering, Design and Selection*, 2(3):193–199, 1988.

Namrata Anand, Raphael Eguchi, Irimpan I Mathews, Carla P Perez, Alexander Derry, Russ B Altman, and Po-Ssu Huang. Protein sequence design with a learned potential. *Nature Communications*, 13(1):1–11, 2022.

Christian B Anfinsen. Principles that govern the folding of protein chains. *Science*, 181 (4096):223–230, 1973.

Frances H Arnold. Directed evolution: Bringing new chemistry to life. *Angewandte Chemie International Edition*, 57(16):4143–4148, 2018.

David Baker. An exciting but challenging road ahead for computational enzyme design. *Protein Science*, 19(10):1817, 2010.

David Baker. Protein folding, structure prediction and design. *Biochemical Society Transactions*, 42(2):225–229, 03 2014. ISSN 0300-5127. doi:10.1042/BST20130055. URL `https://doi.org/10.1042/BST20130055`.

C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

P. Barrat. bmDCA. https://doi.org/10.5281/zenodo.3825613, 2020.

Tristan Bepler and Bonnie Berger. Learning the protein language: Evolution, structure, and function. *Cell Systems*, 12(6):654–669, 2021.

James U Bowie, John F Reidhaar-Olson, Wendell A Lim, and Robert T Sauer. Deciphering the message in protein sequences: tolerance to amino acid substitutions. *Science*, 247 (4948):1306–1310, 1990.

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

Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

François Chollet. Keras: The python deep learning library. Astrophysics Source Code Library (ascl-1806), 2018.

Simona Cocco, Christoph Feinauer, Matteo Figliuzzi, Rémi Monasson, and Martin Weigt. Inverse statistical physics of protein sequences: a key issues review. *Reports on Progress in Physics*, 81(3):032601, 2018.

Qian Cong, Ivan Anishchenko, Sergey Ovchinnikov, and David Baker. Protein interaction networks revealed by proteome coevolution. *Science*, 365(6449):185–189, 2019.

Francis H Crick. On protein synthesis. In *Symposia of the Society for Experimental Biology*, volume 12, pages 138–163. Symposia of the Society for Experimental Biology, 1958.

Vincent Dahirel, Karthik Shekhar, Florencia Pereyra, Toshiyuki Miura, Mikita Artyomov, Shiv Talsania, Todd M. Allen, Marcus Altfeld, Mary Carrington, Darrell J. Irvine, Bruce D. Walker, and Arup K. Chakraborty. Coordinate linkage of HIV evolution reveals regions of immunological vulnerability. *Proceedings of the National Academy of Sciences of the United States of America*, 108(28):11530–11535, 2011. doi:10.1073/pnas.1105315108. URL `https://www.pnas.org/doi/abs/10.1073/pnas.1105315108`.

Bassil I Dahiyat and Stephen L Mayo. De novo protein design: fully automated sequence selection. *Science*, 278(5335):82–87, 1997.

Payel Das, Kahini Wadhawan, Oscar Chang, Tom Sercu, Cicero Dos Santos, Matthew Riemer, Vijil Chenthamarakshan, Inkit Padhi, and Aleksandra Mojsilovic. Pepcvae: Semi-supervised targeted design of antimicrobial peptide sequences. *arXiv preprint arXiv:1810.07743*, 2018.

Rhiju Das and David Baker. Macromolecular modeling with Rosetta. *Annual Review of Biochemistry*, 77(1):363–382, 2008.

Kristian Davidsen, Branden J Olson, William S DeWitt III, Jean Feng, Elias Harkins, Philip Bradley, and Frederick A Matsen IV. Deep generative models for t cell receptor protein sequences. *eLife*, 8:e46935, 2019.

Scott N. Dean and Scott A. Walper. Variational autoencoder for generation of antimicrobial peptides. *ACS Omega*, 5(33):20746–20754, 2020. doi:10.1021/acsomega.0c00442. URL `https://doi.org/10.1021/acsomega.0c00442`. PMID: 32875208.

Nicki Skafte Detlefsen, Søren Hauberg, and Wouter Boomsma. Learning meaningful representations of protein sequences. *Nature Communications*, 13(1):1–12, 2022.

Wenze Ding, Kenta Nakai, and Haipeng Gong. Protein design via deep learning. *Briefings in Bioinformatics*, 23(3):bbac102, 2022.

Xinqiang Ding, Zhengting Zou, and Charles L Brooks III. Deciphering protein evolution and fitness landscapes with latent space models. *Nature communications*, 10(1):1–13, 2019.

Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

Magnus Ekeberg, Cecilia Lovkvist, Yueheng Lan, Martin Weigt, and Erik Aurell. Improved contact prediction in proteins: Using pseudolikelihoods to infer Potts models. *Physical Review E*, 87(1):012707, 2013.

Sibo Feng, James K Chen, Hongtao Yu, Julian A Simon, and Stuart L Schreiber. Two binding orientations for peptides to the src sh3 domain: development of a general model for sh3-ligand interactions. *Science*, 266(5188):1241–1247, 1994.

Andrew L Ferguson and Rama Ranganathan. 100th anniversary of macromolecular science viewpoint: Data-driven protein design. *ACS Macro Letters*, 10(3):327–340, 2021.

Andrew L Ferguson, Jaclyn K Mann, Saleha Omarjee, Thumbi Ndung'u, Bruce D Walker, and Arup K Chakraborty. Translating HIV sequences into quantitative fitness landscapes predicts viral vulnerabilities for rational immunogen design. *Immunity*, 38(3):606–617, 2013.

Peter Fields, Vudtiwat Ngampruetikorn, Rama Ranganathan, David Schwab, and Stephanie Palmer. Finding the function-determining subset of amino acids in protein sequence data. *Bulletin of the American Physical Society*, 2023.

Matteo Figliuzzi, Pierre Barrat-Charlaix, and Martin Weigt. How pairwise coevolutionary models capture the collective residue variability in proteins? *Molecular Biology and Evolution*, 35(4):1018–1027, 2018.

Vincent Frappier and Amy E Keating. Data-driven computational protein design. *Current Opinion in Structural Biology*, 69:63–69, 2021.

Chase R Freschlin, Sarah A Fahlberg, and Philip A Romero. Machine learning to navigate fitness landscapes for protein engineering. *Current Opinion in Biotechnology*, 75:102713, 2022.

Wenhao Gao, Sai Pooja Mahajan, Jeremias Sulam, and Jeffrey J Gray. Deep learning in protein structural modeling and design. *Patterns*, 1(9):100142, 2020.

Andrew Giessel, Athanasios Dousis, Kanchana Ravichandran, Kevin Smith, Sreyoshi Sur, Iain McFadyen, Wei Zheng, and Stuart Licht. Therapeutic enzyme engineering using a generative neural network. *Scientific Reports*, 12(1):1–17, 2022.

R Daniel Gietz and Robert H Schiestl. High-efficiency yeast transformation using the LiAc/SS carrier DNA/PEG method. *Nature Protocols*, 2(1):31–34, 2007.

Ulrike Göbel, Chris Sander, Reinhard Schneider, and Alfonso Valencia. Correlated mutations and residue contacts in proteins. *Proteins: Structure, Function, and Bioinformatics*, 18 (4):309–317, 1994.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

Joe G Greener, Lewis Moffat, and David T Jones. Design of metalloproteins and novel protein folds using variational autoencoders. *Scientific Reports*, 8(1):1–12, 2018.

Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample-problem. *Advances in Neural Information Processing Systems*, 19:513–520, 2006.

Xiaojie Guo, Sivani Tadepalli, Liang Zhao, and Amarda Shehu. Generating tertiary protein structures via an interpretative variational autoencoder. *arXiv preprint arXiv:2004.07119*, 2020.

Najeeb Halabi, Olivier Rivoire, Stanislas Leibler, and Rama Ranganathan. Protein sectors: Evolutionary units of three-dimensional structure. *Cell*, 138(4):774–786, 2009.

Harris Sarah Harris David. *Digital Design and Computer Architecture (2nd ed.).* Morgan Kaufmann, 2013.

Gregory R Hart and Andrew L Ferguson. Empirical fitness models for hepatitis C virus immunogen design. *Physical Biology*, 12(6):066006, 2015.

Mohamad H Hassoun. *Fundamentals of Artificial Neural Networks.* MIT Press, 1995.

Alex Hawkins-Hooker, Florence Depardieu, Sebastien Baur, Guillaume Couairon, Arthur Chen, and David Bikard. Generating functional protein variants with variational autoencoders. *PLoS Computational Biology*, 17(2):e1008736, 2021.

Thomas A Hopf, John B Ingraham, Frank J Poelwijk, Charlotta PI Schärfe, Michael Springer, Chris Sander, and Debora S Marks. Mutation effects predicted from sequence co-variation. *Nature Biotechnology*, 35(2):128–135, 2017.

Po-Ssu Huang, Scott E Boyken, and David Baker. The coming of age of de novo protein design. *Nature*, 537(7620):320–327, 2016.

John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. Generative models for graph-based protein design. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/f3a4ff4839c56a5f460c88cce3666a2b-Paper.pdf`.

Christian Jäckel, Peter Kast, and Donald Hilvert. Protein design by directed evolution. *Annual Review of Biochemistry*, 37:153–173, 2008.

Neil P King, William Sheffler, Michael R Sawaya, Breanna S Vollmar, John P Sumida, Ingemar André, Tamir Gonen, Todd O Yeates, and David Baker. Computational design of self-assembling protein nanomaterials with atomic level accuracy. *Science*, 336(6085): 1171–1174, 2012.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Marc Kirschner and John Gerhart. Evolvability. *Proceedings of the National Academy of Sciences*, 95(15):8420–8427, 1998.

Gert Kiss, Nihan Çelebi-Ölçüm, Rocco Moretti, David Baker, and KN Houk. Computational enzyme design. *Angewandte Chemie International Edition*, 52(22):5700–5725, 2013.

Yaakov Kleeorin, William P. Russ, Olivier Rivoire, and Rama Ranganathan. Undersampling and the inference of coevolution in proteins. *bioRxiv*, 2021. doi:10.1101/2021.04.22.441025. URL `https://www.biorxiv.org/content/early/2021/04/23/2021.04.22.441025`.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 14, pages 1137–1145, 1995.

Brian Kuhlman, Gautam Dantas, Gregory C Ireton, Gabriele Varani, Barry L Stoddard, and David Baker. Design of a novel globular protein fold with atomic-level accuracy. *Science*, 302(5649):1364–1368, 2003.

Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference* *on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1718–1727, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/li15.html`.

Xinran Lian, Nikša Praljak, Subu Subramanian, Sarah Wasinger, Rama Ranganathan, and Andrew L Ferguson. Deep learning-enabled design of synthetic orthologs of a signaling protein. *bioRxiv*, page doi: 10.1101/2022.12.21.521443, 2022. URL `https://doi.org/10.1101/2022.12.21.521443`.

Wendell A Lim, Robert O Fox, and Frederic M Richards. Stability and peptide binding affinity of an sh3 domain from the caenorhabditis elegans signaling protein sem-5. *Protein Science*, 3(8):1261–1266, 1994.

Jin Liu and Ruth Nussinov. Allostery: An overview of its history, concepts, methods, and applications. *PLoS Computational Biology*, 12(6):e1004966, 2016.

Andrei N Lupas, Joana Pereira, Vikram Alva, Felipe Merino, Murray Coles, and Marcus D Hartmann. The breakthrough in protein structure prediction. *Biochemical Journal*, 478 (10):1885–1890, 2021.

Ali Madani, Ben Krause, Eric R. Greene, Subu Subramanian, Benjamin P. Mohr, James M. Holton, Jose Luis Olmos, Caiming Xiong, Zachary Z. Sun, Richard Socher, James S. Fraser, and Nikhil Naik. Deep neural language modeling enables functional protein generation across families. *bioRxiv*, page 2021.07.18.452833, 2021. doi:10.1101/2021.07.18.452833.

Jaclyn K Mann, John P Barton, Andrew L Ferguson, Saleha Omarjee, Bruce D Walker, Arup Chakraborty, and Thumbi Ndung'u. The fitness landscape of HIV-1 gag: Advanced modeling approaches and validation of model predictions by in vitro testing. *PLoS Computational Biology*, 10(8):e1003776, 2014.

Jennifer A Marles, Samira Dahesh, Jennifer Haynes, Brenda J Andrews, and Alan R Davidson. Protein-protein interaction affinity plays a crucial role in controlling the Sho1p-mediated signal transduction pathway in yeast. *Molecular Cell*, 14(6):813–823, 2004. ISSN 1097-2765. doi:https://doi.org/10.1016/j.molcel.2004.05.024. URL `https://www.sciencedirect.com/science/article/pii/S1097276504003260`.

Bruce J Mayer. Sh3 domains: complexity in moderation. *Journal of cell science*, 114(7): 1253–1263, 2001.

John Maynard Smith. Natural selection and the concept of a protein space. *Nature*, 225 (5232):563–564, 1970.

Stanislav Mazurenko, Zbynek Prokop, and Jiri Damborsky. Machine learning in enzyme engineering. *ACS Catalysis*, 10(2):1210–1223, 2019.

Conor J McClune, Aurora Alvarez-Buylla, Christopher A Voigt, and Michael T Laub. Engineering orthogonal signalling pathways reveals the sparse occupancy of sequence space. *Nature*, 574(7780):702–706, 2019.

Faruck Morcos, Andrea Pagnani, Bryan Lunt, Arianna Bertolino, Debora S Marks, Chris Sander, Riccardo Zecchina, José N Onuchic, Terence Hwa, and Martin Weigt. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences of the United States of America*, 108(49):E1293–E1301, 2011.

Andrea Musacchio, Martin Noble, Richard Pauptit, Rik Wierenga, and Matti Saraste. Crystal structure of a Src-homology 3 (SH3) domain. *Nature*, 359(6398):851–855, 1992.

Margarita Osadchy and Rachel Kolodny. How deep learning tools can help protein engineers find good sequences. *The Journal of Physical Chemistry B*, 125(24):6440–6450, 2021.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

Jimin Pei, Bong-Hyun Kim, and Nick V Grishin. PROMALS3D: A tool for multiple protein sequence and structure alignments. *Nucleic Acids Research*, 36(7):2295–2300, 2008.

Mark E Peterson, Feng Chen, Jeffery G Saven, David S Roos, Patricia C Babbitt, and Andrej Sali. Evolutionary constraints on structural similarity in orthologs and paralogs. *Protein Science*, 18(6):1306–1315, 2009.

Francesc Posas and Haruo Saito. Osmotic activation of the HOG MAPK pathway via Ste11p MAPKKK: Scaffold role of Pbs2p MAPKK. *Science*, 276(5319):1702–1705, 1997.

Niksa Praljak and Andrew Ferguson. Auto-regressive wavenet variational autoencoders for alignment-free generative protein design and fitness prediction. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

Niksa Praljak, Xinran Lian, Rama Ranganathan, and Andrew Ferguson. Protwave-vae: Integrating autoregressive sampling with latent-based inference for data-driven protein design. *bioRxiv*, pages 2023–04, 2023.

Donatas Repecka, Vykintas Jauniskis, Laurynas Karpus, Elzbieta Rembeza, Irmantas Rokaitis, Jan Zrimec, Simona Poviloniene, Audrius Laurynenas, Sandra Viknander, Wissam Abuajwa, Otto Savolainen, Rolandas Meskys, Martin K. M. Engqvist, and Aleksej Zelezniak. Expanding functional protein sequence spaces using generative adversarial networks. *Nature Machine Intelligence*, 3(4):324–333, 2021.

Kimberly A Reynolds, William P Russ, Michael Socolich, and Rama Ranganathan. Evolution-based design of proteins. In *Methods in Enzymology*, volume 523, pages 213–235. Elsevier, 2013.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

Danilo Jimenez Rezende and Fabio Viola. Taming VAEs. *arXiv preprint arXiv:1810.00597*, 2018.

Adam J Riesselman, John B Ingraham, and Debora S Marks. Deep generative models of genetic variation capture the effects of mutations. *Nature Methods*, 15(10):816–822, 2018.

Adam J Riesselman, Jung-Eun Shin, Aaron W Kollasch, Conor McMahon, Elana Simon, Chris Sander, Aashish Manglik, Andrew C Kruse, and Debora S Marks. Accelerating protein design using autoregressive generative models. *bioRxiv*, page 757252, 2019.

Olivier Rivoire, Kimberly A Reynolds, and Rama Ranganathan. Evolution-based functional decomposition of proteins. *PLoS Computational Biology*, 12(6):e1004817, 2016.

Philip A Romero and Frances H Arnold. Exploring protein fitness landscapes by directed evolution. *NatureRreviews Molecular Cell Biology*, 10(12):866–876, 2009.

Philip A Romero, Andreas Krause, and Frances H Arnold. Navigating the protein fitness landscape with gaussian processes. *Proceedings of the National Academy of Sciences of the United States of America*, 110(3):E193–E201, 2013.

William P Russ, Drew M Lowery, Prashant Mishra, Michael B Yaffe, and Rama Ranganathan. Natural-like function in artificial ww domains. *Nature*, 437(7058):579–583, 2005.

William P. Russ, Matteo Figliuzzi, Christian Stocker, Pierre Barrat-Charlaix, Michael Socolich, Peter Kast, Donald Hilvert, Remi Monasson, Simona Cocco, Martin Weigt, and Rama Ranganathan. An evolution-based model for designing chorismate mutase enzymes. *Science*, 369(6502):440–445, 2020. doi:10.1126/science.aba3304. URL `https://www.science.org/doi/abs/10.1126/science.aba3304`.

Kalle Saksela and Perttu Permi. Sh3 domain ligand binding: What's the consensus and where's the specificity? *FEBS letters*, 586(17):2609–2614, 2012.

Jörn M Schmiedel and Ben Lehner. Determining protein structures using deep mutagenesis. *Nature Genetics*, page 1, 2019.

Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi:10.1002/j.1538-7305.1948.tb01338.x.

Karthik Shekhar, Claire F Ruberman, Andrew L Ferguson, John P Barton, Mehran Kardar, and Arup K Chakraborty. Spin models inferred from patient-derived viral sequence data faithfully describe HIV fitness landscapes. *Physical Review E*, 88(6):062705, 2013.

Jung-Eun Shin, Adam J Riesselman, Aaron W Kollasch, Conor McMahon, Elana Simon, Chris Sander, Aashish Manglik, Andrew C Kruse, and Debora S Marks. Protein design and variant prediction using autoregressive generative models. *Nature Communications*, 12(1):1–11, 2021.

Robert S Sikorski and Philip Hieter. A system of shuttle vectors and yeast host strains designed for efficient manipulation of dna in saccharomyces cerevisiae. *Genetics*, 122(1): 19–27, 1989.

Sam Sinai, Eric Kelsic, George M Church, and Martin A Nowak. Variational auto-encoding of protein sequences. *arXiv preprint arXiv:1712.03346*, 2017.

Sam Sinai, Nina Jain, George M Church, and Eric D Kelsic. Generative AAV capsid diversification by latent interpolation. *bioRxiv*, page 2021.04.16.440236, 2021. doi:10.1101/2021.04.16.440236. URL `https://www.biorxiv.org/content/early/2021/04/17/2021.04.16.440236`.

Michael Socolich, Steve W Lockless, William P Russ, Heather Lee, Kevin H Gardner, and Rama Ranganathan. Evolutionary information for specifying a protein fold. *Nature*, 437 (7058):512–518, 2005.

Michael A Stiffler, Doeke R Hekstra, and Rama Ranganathan. Evolvability as a function of purifying selection in tem-1 β-lactamase. *Cell*, 160(5):882–892, 2015.

Subramanian Kanagarajan Subramanian. *Distinct functional phases in proteins: A test by large-scale protein design.* PhD thesis, UT Southwestern Medical Center, 2017.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf`.

William R Taylor. A 'periodic table'for protein structures. *Nature*, 416(6881):657–660, 2002.

Pengfei Tian, John M Louis, James L Baber, Annie Aniana, and Robert B Best. Co-evolutionary fitness landscapes for sequence design. *Angewandte Chemie International Edition*, 57(20):5674–5678, 2018.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.

Jing-Ke Weng. The evolutionary paths towards complexity: a metabolic perspective. *New Phytologist*, 201(4):1141–1149, 2014.

Timothy A Whitehead, Aaron Chevalier, Yifan Song, Cyrille Dreyfus, Sarel J Fleishman, Cecilia De Mattos, Chris A Myers, Hetunandan Kamisetty, Patrick Blair, Ian A Wilson, and David Baker. Optimization of affinity, specificity and function of designed influenza inhibitors using deep sequencing. *Nature Biotechnology*, 30(6):543–548, 2012.

Bruce J Wittmann, Kadina E Johnston, Zachary Wu, and Frances H Arnold. Advances in machine learning for directed evolution. *Current Opinion in Structural Biology*, 69:11–18, 2021.

Yuting Xu, Deeptak Verma, Robert P Sheridan, Andy Liaw, Junshui Ma, Nicholas M Marshall, John McIntosh, Edward C Sherer, Vladimir Svetnik, and Jennifer M Johnston. Deep dive into machine learning models for protein engineering. *Journal of Chemical Information and Modeling*, 60(6):2773–2790, 2020.

Kevin K Yang, Zachary Wu, and Frances H Arnold. Machine-learning-guided directed evolution for protein engineering. *Nature Methods*, 16(8):687–694, 2019.

Ali Zarrinpar, Sang-Hyun Park, and Wendell A Lim. Optimization of specificity in a cellular protein interaction network by negative selection. *Nature*, 426(6967):676–680, 2003.

Cathleen Zeymer and Donald Hilvert. Directed evolution of protein catalysts. *Annual review of biochemistry*, 87:131–157, 2018.

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Information maximizing variational autoencoders. *arXiv preprint arXiv:1706.02262*, 2017.

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Balancing learning and inference in variational autoencoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5885–5892, 2019.