Supporting Information

De Novo Protein Design for Novel Folds Using Guided Conditional Wasserstein Generative Adversarial Networks

Mostafa Karimi†, Shaowen Zhu†, Yue Cao†, and Yang Shen*
Department of Electrical and Computer Engineering
Texas A&M University
College Station, TX 77843-3128, USA
†: Co-First Authors
*: Correspondence: yshen@tamu.edu

1 Data

1.1 Pre-processing

We download fold data (including corresponding sequences and structure domains) from SCOPe (v. 2.07) and filtered sequences at 100% identity level. Some uncommon sequences (<2%) contain non-standard amino acid letters including 'b', 'z', 'x' and 'X'. The first three represent the ambiguity among ['d', 'n'], ['q','e'] and all 20 amino acids, respectively; and 'X' represents a gap in sequence. For simplicity, we assign explicit standard amino acids to each occasion of 'b', 'z', and 'x' following a uniform prior and disregard sequences containing 'X'.

To save the computational cost under limited budget, we further filter the remaining and retained those of sequence length between 60 and 160. The length interval is chosen to balance the range (for sequence padding concerns) and the fold space coverage. Specifically, a tight range around the mode of the distribution would lead to sequences of similar lengths and reduce padding needs. Meanwhile, the range needs to be wide enough to cover the sequence and the fold space. In the end, the chosen range represents over 35% of the filtered sequences (Fig. S1) and over 63% of the folds (781 out of 1,232).

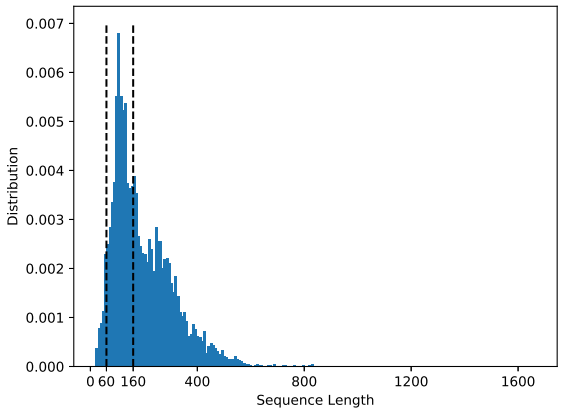


Figure S1: Sequence length distribution in the SCOPe data (100% sequence identity level). The percentage within the interval [60, 160] is 35.87%.

1.2 Statistics

SCOPe clustered the sequences into 7 classes indexed from a to g with names and counts given in Table S1. We also classify the folds into three difficulty categories based on sequence abundance: easy for those with more than 50 sequences, medium for 6 to 50 sequences, and difficult for at most 5 sequences. The split counts based on the combination of fold class and difficulty are provided in Table S1.

Class Index	Class Name	Easy	Medium	Difficulty	Total
a	α	27	64	105	196
b	β	28	57	54	139
c	α/β	15	35	18	68
d	$\alpha+\beta$	38	110	140	288
e	Multi-domain proteins	0	0	6	6
f	Membrane and cell surface	2	9	10	21
g	Small proteins	7	17	39	63
	Total	117	292	372	781

Table S1: The statistic of folds in the whole dataset based on class and difficulty.

We use stratified sampling to split the resulting dataset into training (70%), validation (15%) and test (15%) sets while preserving the fold-class distributions.

Class Index	Class Name	Training Set	Val. Set	Test Set	Total
a	α	139	29	28	196
b	β	109	15	15	139
c	α/β	52	8	8	68
d	$\alpha+\beta$	206	41	41	288
e	Multi-domain proteins	3	2	1	6
f	Membrane and cell surface	14	4	3	21
g	Small proteins	40	12	11	63
	Total	563	111	107	781

Table S2: The statistics of folds in training, validation and testing sets based on class.

Class Index	Class Name	# of Seq.
a	α	5486
b	β	4562
c	α/β	2536
d	$\alpha+\beta$	6625
e	Multi-domain proteins	8
f	Membrane and cell surface	189
g	Small proteins	719
	Total	20125

Table S3: The statistics of sequences in training set.

Sequence representative(s) of each fold are chosen for post-analysis. With pairwise sequence identities as similarity measure, all sequences of the same fold are clustered using DBSCAN (Ester et al., 1996) ($\epsilon = 0.7$ as proteins of sequence identity over 0.3 very likely have similar structures). A representative is chosen for each cluster based on the maximum sum of sequence identities towards all other sequences within the same cluster.
A single structure representative is already chosen for each fold by SCOPe.

2 Conditional Input: Fold representation

2.1 Methods

We assume that the fold space has a very high dimensionality, where each of 1,232 existing folds (like a point) lies. Without specifically describing each fold, we describe distances between fold pairs (using pairwise fold similarity) and then try to find the first K orthonormal axes (principal components) spanning a subspace so that the 1,232 data points have the largest variance in that space. These K principal components form a K-dimensional subspace. For any new fold, we project it into the subspace (specifically, onto its principal components) to get K coordinates, which is our final fold representation.

Specifically, using a representative protein structure chosen by SCOPe for each of the 1,232 folds, we construct a pairwise similarity matrix of symmetrized TM-scores (Zhang and Skolnick, 2004). The TM-score measures the similarity between two folds in the space. Since TM-score is not symmetric, for each pair of two structures, we first choose one as the target and the other as the reference in TM-align (Zhang and Skolnick, 2005) and calculate the TM-scores (normalized by the length of the second structure as the reference). Then we swap the target and the reference in TM-align and calculate a TM-score again. The average of the two TM-scores are referred to as the symmetric TM-score.

Starting with the matrix of symmetrized TM-scores, we use (Higham, 2002) to compute the nearest positive-definite Gram matrix. We centralize the Gram matrix $\boldsymbol{S}$ through:

$$\begin{aligned} \boldsymbol{O}_{\mathrm{NN}} &= \mathbf{1}_{\mathrm{N}}\mathbf{1}_{\mathrm{N}}^{T}/N; \\ \tilde{\boldsymbol{S}} &= (\boldsymbol{I} - \boldsymbol{O}_{\mathrm{NN}})\boldsymbol{S}(\boldsymbol{I} - \boldsymbol{O}_{\mathrm{NN}}); \end{aligned} \tag{1}$$

where N being 1,232 is the row and column size of $\boldsymbol{S}$, $\mathbf{1}_{\mathrm{N}}$ is an all-one column vector of size N by 1, and $\boldsymbol{I}$ is an identity matrix of size N by N. We then perform eigen-decomposition to $\tilde{\boldsymbol{S}}$ and rank its eigenvalues u_i and its corresponding eigenvectors $\boldsymbol{v}_i$ in a decreasing order: $\{(u_1, \boldsymbol{v}_1), (u_2, \boldsymbol{v}_2), ..., (u_N, \boldsymbol{v}_N)\}$. The ith dimension basis of fold space is calculated through:

$$\boldsymbol{r}_i = \frac{\boldsymbol{v}_i}{\sqrt{u_i}} \tag{2}$$

Then for M incoming new folds, we first calculate their TM-scores with individual N basis folds and form a new matrix $\boldsymbol{S}^*$ with size $M \times N$. We then centralize $\boldsymbol{S}^*$ though

$$\begin{aligned} \boldsymbol{O}_{\mathrm{MN}} &= \mathbf{1}_{\mathrm{M}}\mathbf{1}_{\mathrm{N}}^{T}/N; \\ \tilde{\boldsymbol{S}}^* &= \boldsymbol{S}^* - \boldsymbol{S}^*\boldsymbol{O}_{\mathrm{NN}} - \boldsymbol{O}_{\mathrm{MN}}\boldsymbol{S} + \boldsymbol{O}_{\mathrm{MN}}\boldsymbol{S}\boldsymbol{O}_{\mathrm{NN}}; \end{aligned} \tag{3}$$

And then the coordinate vector of these these M folds along the ith dimension is $\tilde{\boldsymbol{S}}^* \cdot \boldsymbol{r}_i$.

2.2 Results

We first examine how much of the variance in the original fold space spanned by 1,232 basis folds can be explained by the top eigenvectors from kernel PCA (kPCA). As seen in Fig. S2, the first 20 eigenvectors explained 14.8% of the variance in the original fold space spanned by 1,232 basis folds and the first 200 did over 50%..

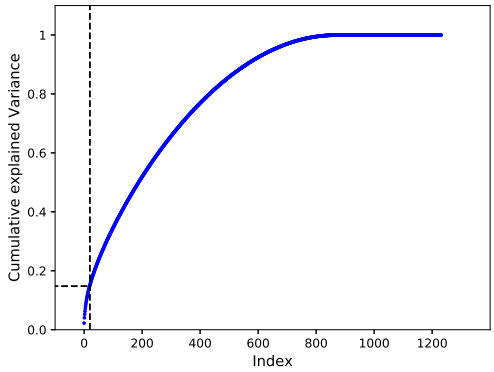


Figure S2: Cumulative variance explained by the top eigenvectors of kernel PCA for fold representation.

Due to the computational limit of deep generative models, we have to restrict the fold representation to a space spanned by the 20 eigenvectors. In this case, we ask what resolution of the fold space is accomplished by the representation. To answer the question, we cluster these 1,232 folds into varying K clusters through K-means clustering. The centriods of these K clusters form a K-dimensional subspace for the original 1,232-dimensional space. We then project the aforementioned 20 eigenvectors into this subspace and calculate the explained variance of those 20 projected vectors for the subspace. As seen in Fig. S3, the 20 eigenvectors could explain over 60% of the variance in a fold subspace spanned by 40 representative folds.

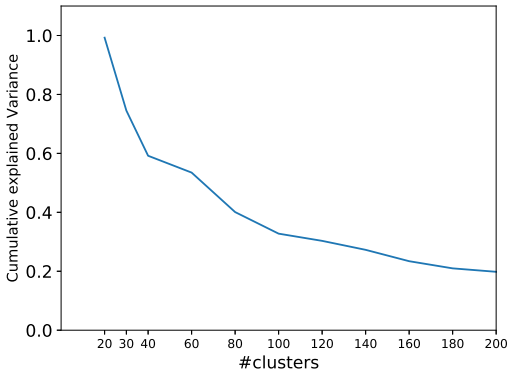


Figure S3: The explained variance of the K clusters (or the K-dimensional subspace subspace of the 1232D original fold space) using the aforementioned 20 eigenvectors.

The new representations of 1,232 folds are visualized in a 2D space using t-SNE. Fig. S4.

3 Oracle: Protein Fold Prediction

3.1 Methods

An oracle of protein fold recognition for a query sequence is needed for both guiding sequence generation during model training and assessing accuracy once the model is trained. Since there is no closed-form mathematical formula for protein fold as a function of the sequence, protein fold is often predicted through data-driven approaches such as DeepSF (Hou et al., 2017). Hou and co-workers designed a 1D deep convolutional neural network to classify protein folds with variable-length to each of 1,195 protein folds. The input features include 1) amino acid (AA) sequences, 2) position-specific scoring matrix (PSSM), 3) predicted secondary

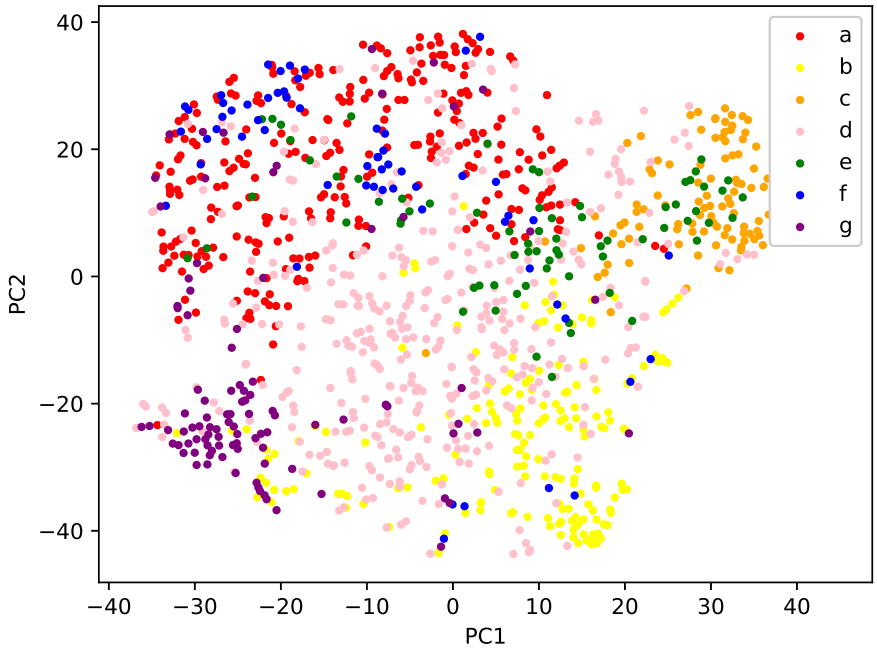


Figure S4: 2D t-SNE visualization of 1,232 folds in the resulting 20D representation. Folds are colored based on fold classes as follows. a: α; b: β; c: α/β; d: $\alpha + \beta$; e: α and β; f: membrane and cell surface; and g: small proteins.

structure (SS), and 4) predicted solvent accessibility (SA). PSI-BLAST for multiple sequence alignment is required for the calculation of PSSM, SS and SA. PSI-BLAST costs minutes for each sequence and can be tolerated for one-time feature generation for few sequences. However, it is computationally daunting for on-line feature calculation for the huge number of sequences generated during our model training.
Therefore, we modified DeepSF to classify a protein into 1,215 folds (slightly increased to consider SCOPe update) using one-hot encoding of sequence alone. Since sequence features were found the least informative, such a model would significantly suffer in accuracy. We thus further modified the architecture of DeepSF neural networks to be wider and deeper (with residual blocks). Specifically, we increase the filter size from 10 to 40; and deepen the model from 10 layers of 1D convolution to 20 layers by inserting 10 residual convolutional layers. Modified DeepSF architecture is shown as part of Fig. S5.
Modified DeepSF consists of two branches with filter sizes of 6 and 10 to capture short and long motif information from the sequence, respectively. Each branch consists of blocks of convolution layers followed by batch normalization and ReLU activation. These blocks are non-residual at the beginning and at the end, but residual (connected through skip connections) in the middle. At the end of each branch, a customized max pooling layer has been implemented to capture only the top m amino acids from the sequence. This max pooling is required to deal with the length varying sequences so that its output is of fixed length m (In our model m is equal to 30). Lastly, considering the limited accuracy of the top-1 fold predictions, we further allow the ambiguity of the oracle and use its top-10 predictions to guide or assess each generated sequence.

3.2 Results

We compare the accuracy for fold prediction between the original and the modified DeepSF in Table S4. The model trained for the 1,215 folds using sequence only is adopted as the oracle.

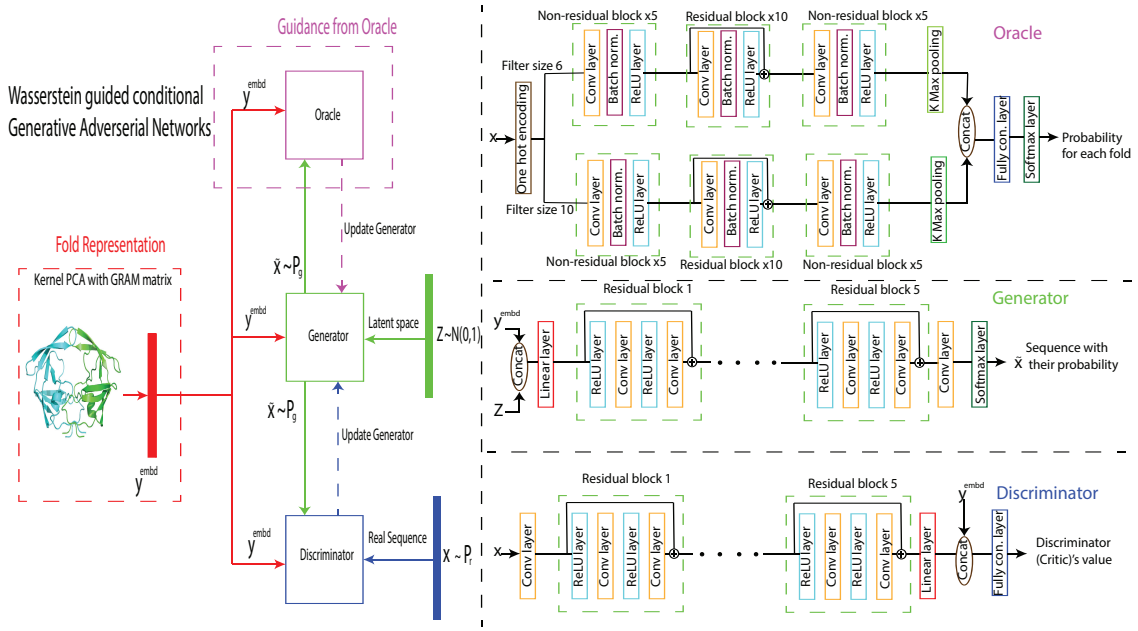


Figure S5: The overall architecture of our guided conditional Wasserstein GAN (gcWGAN) including those of the oracle, the generator, and the discriminator.

Prediction	Training set				Test set			
	Original DeepSF		Modified DeepSF		Original DeepSF		Modified DeepSF	
	All features (1,195 folds)	AA only (1,195 folds)	AA only (1,195 folds)	AA only (1,215 folds)	All features (1,195 folds)	AA only (1,195 folds)	AA only (1,195 folds)	AA only (1,215 folds)
Top1	0.74	0.36	0.56	0.49	0.72	0.33	0.40	0.36
Top5	0.93	0.65	0.82	0.75	0.9	0.59	0.63	0.61
Top10	0.97	0.76	0.89	0.84	0.94	0.69	0.74	0.72
Top15	0.98	0.82	0.92	0.88	0.96	0.74	0.79	0.78
Top20	0.98	0.86	0.94	0.91	0.97	0.79	0.82	0.81

Table S4: Fold prediction accuracy for the original and our modified DeepSF models with various features and architectures.

4 GAN Models for De Novo Protein Design

Generative Adversarial Network (GAN) (Goodfellow et al., 2014), a very successful and powerful class of generative models, represents a game between a generator G and a discriminator D. The generator's objective is to generates artificial data from a noise input that are close to real data and the discriminator's goal is to discriminates the generated data from the real ones. GAN has wide applications such as generating images of unprecedented quality of celebrities (Karras et al., 2017), image to image translation (Chu et al., 2017), text to image translation (Zhang et al., 2017), creating anime characters (Jin et al., 2017), image inpainting (Pathak et al., 2016), face aging (Antipov et al., 2017) and music generation (Yang et al., 2017). It has also been extended from an unsupervised generative model to a supervised one through conditional GAN (Mirza and Osindero, 2014) that generates images specific for each class.

Despite their popularity, GANs are notoriously hard to train due to problems such as the difficulty to achieve Nash equilibrium (Salimans et al., 2016), low dimensional support, vanishing gradient and mode collapsing (Arjovsky et al.). Wasserstein GAN (WGAN) (Arjovsky et al.; Gulrajani et al., 2017) is a fundamental improvement to the original GANs that addresses these problems. WGAN replaces the KL-divergence with Wasserstein (or earth-mover's) distance. The latter has less issues toward low dimensional support (thus more stable) and use the notation critics instead of the discriminator.

4.1 Conditional WGAN

See **Algorithm 1**

Algorithm 1 Conditional WGAN with gradient penalty

Randomly initialize the parameter from scratch
while θ has not converged **do**
  for t=1$\cdots n_{\text{critic}}$ **do**
    Sample real data $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$
    Sample noise $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p(\mathbf{z})$
    Sample random number $\{\epsilon^{(i)}\}_{i=1}^{m} \sim U[0,1]$
    $\{\tilde{\mathbf{x}}^{(i)}\}_{i=1}^{m} \leftarrow \{G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{(i)})\}_{i=1}^{m}$
    $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^{m} \leftarrow \{\epsilon^{(i)}\mathbf{x}^{(i)} + (1-\epsilon^{(i)})\tilde{\mathbf{x}}^{(i)}\}_{i=1}^{m}$
    $L \leftarrow \frac{1}{m}\sum_{i=1}^{m} D_w(\mathbf{x}^{(i)}|\mathbf{y}^{(i)}) - D_w(\tilde{\mathbf{x}}^{(i)}|\mathbf{y}^{(i)}) - \lambda(||\nabla_{\hat{\mathbf{x}}^{(i)}} D_w(\hat{\mathbf{x}}^{(i)}|\mathbf{y}^{(i)})||_2 - 1)^2$
    $w \leftarrow \text{Adam}(\nabla_w L, w, \alpha, \beta_1, \beta_2)$
  Sample a batch of noises $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p(\mathbf{z})$
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} -D_w(G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{(i)})), \theta, \alpha, \beta_1, \beta_2)$

4.2 Guided cWGAN (gcWGAN)

See **Algorithm 2**

Algorithm 2 guided cWGAN with gradient penalty

Initialize the parameter from the previously trained semi-supervised cWGAN
while θ has not converged **do**
  for t=1$\cdots$,n_{critic} **do**
    Sample real data $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$
    Sample noise $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p(\mathbf{z})$
    Sample random number $\{\epsilon^{(i)}\}_{i=1}^{m} \sim U[0,1]$
    $\{\tilde{\mathbf{x}}^{(i)}\}_{i=1}^{m} \leftarrow \{G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{(i)})\}_{i=1}^{m}$
    $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^{m} \leftarrow \{\epsilon^{(i)}\mathbf{x}^{(i)} + (1-\epsilon^{(i)})\tilde{\mathbf{x}}^{(i)}\}_{i=1}^{m}$
    $L_D \leftarrow \frac{1}{m}\sum_{i=1}^{m} D_w(\mathbf{x}^{(i)}|\mathbf{y}^{(i)}) - D_w(\tilde{\mathbf{x}}^{(i)}|\mathbf{y}^{(i)}) - \lambda_1(||\nabla_{\hat{\mathbf{x}}^{(i)}} D_w(\hat{\mathbf{x}}^{(i)}|\mathbf{y}^{(i)})||_2 - 1)^2$
    $w \leftarrow \text{Adam}(\nabla_w L_D, w, \alpha, \beta_1, \beta_2)$
  Sample a batch of noises $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p(\mathbf{z})$
  Sample real data $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$
  $\{\tilde{\mathbf{x}}^{(i)}\}_{i=1}^{m} \leftarrow \{G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{(i)})\}_{i=1}^{m}$
  $L_G \leftarrow \frac{1}{m}\sum_{i=1}^{m} -D_w(\tilde{\mathbf{x}}^{(i)}) + \lambda_2[I_{\text{target}}(\tilde{\mathbf{x}}^{(i)}) \log O(\tilde{\mathbf{x}}^{(i)}) + (1 - I_{\text{target}}(\tilde{\mathbf{x}}^{(i)})) \log(1 - O(\tilde{\mathbf{x}}^{(i)})]$
  $\theta \leftarrow \text{Adam}(\nabla_\theta L_G, \theta, \alpha, \beta_1, \beta_2)$

4.3 Architecture of GAN Models

For all the three GAN models (cWGAN, semi-supervised cWGAN, and semi-supervised gcWGAN) we use the same architecture for both the generator and the discriminator, as inspired by WGAN-GP (Gulrajani et al., 2017). The generator and the discriminator's architectures in GAN are usually similar in computational expressiveness so that the training process could be stable, thus they are often transpose of each other. Both consist of residual blocks which include 2 layers of RELU for nonlinearity and 1D convolution with the skip connection between the input and the output.

As shown in the middle-right panel of Fig. S5, noise $\boldsymbol{z}$'s distribution as latent sequence space and fold representation $\boldsymbol{y}$ as the conditioning part are the inputs to the generator. Let L be the maximal sequence length. The generator's architecture consists of the concatenation of $\boldsymbol{z}$ and $\boldsymbol{y}$ followed by a linear mapping to a higher-dimensional space ($L \times 500$). Five layers of residual blocks in sequence (20 layers in total) help

improve the expressiveness of the model and their skip connections help avoid the gradient vanishing problem. Finally, 1D convolution brings down the dimensionality from $L \times 500$ to $L \times 21$ (20 characters for amino acids and one for padding); and a softmax layer produces a probability distribution over the characters for each position. The output of the generator is a continuous distribution over amino acids for each position. It can be converted to a discrete sequence by considering the character with the maximum probability for each position.
The generator's continuous distribution over the amino acids for each position with the fold representation are the inputs to the discriminator. The continuous distribution is used as the approximation for the one-hot encoding of discrete sequences because the latter, being non-differentiable, causes challenges in gradient calculation. First, a 1D convolutional layer converts the continuous distribution over amino acids ($L \times 21$) to a high dimension ($L \times 500$), followed by five layers of residual blocks to make the model deeper. Next, a linear layer brings the dimensionality down to 100. It is then concatenated with the fold representation and follow by the a fully connected layer with 300 neurons. Finally, a linear model converts the 300 dimensional space down to 1 dimension which is the output of the discriminator (or the critic in the WGAN).

5 Hyper-Parameter Tuning

5.1 Criteria

5.1.1 Convergence

The generator in cWGAN tries to minimize the loss function defined in Eq. 1 in the main text and the discriminator tries to maximize the loss. This min-max game originates from the zero-sum game in game theory. Since the generator seeks to minimize the loss, we can compare generators based on the total loss and the those with lower losses are more desirable. In WGAN, the total loss is named as "critic loss" since 1) the discriminator is named as the critic in WGAN and 2) the total loss is exactly the discriminator's loss when we train the model (notice that the discriminator is present in all the three terms of L_2; however, the generator is only present in the second term).
To find out if the model has reached mathematical convergence, we assess the critic's loss over hundreds of epochs.

5.1.2 Nonsense Sequence Ratio

In our model, the length of the output of the generator or the input of the discriminator is fixed, but the length of the real sequences can be varied. So we fixed the output or input length to be the longest possible value (160 in the current study). And we used padding to fill the entries behind the sequence if its length is smaller than that value. One problem is that in the generated sequences, there can be padding appearing between or in front of the residue characters, which make it not resemble real sequences. We regarded these sequences as invalid or nonsense sequences, and called the ratio of the invalid ones in all generated sequences as the nonsense sequence ratio. Nonsense ratio was taken as one criteria to show the model performance in biological relevance. And for the generated sequences, firstly they need pass the sanity check; only those that pass the check are fed to the next-step and the nonsense sequences are discarded.

5.1.3 Padding Ratio

A sequence that passes nonsense check above can still contain too much paddings to be relevant. We thus calculate the ratio of paddings over the whole sequence of characters and monitor them during model training.

5.1.4 Sequence Novelty

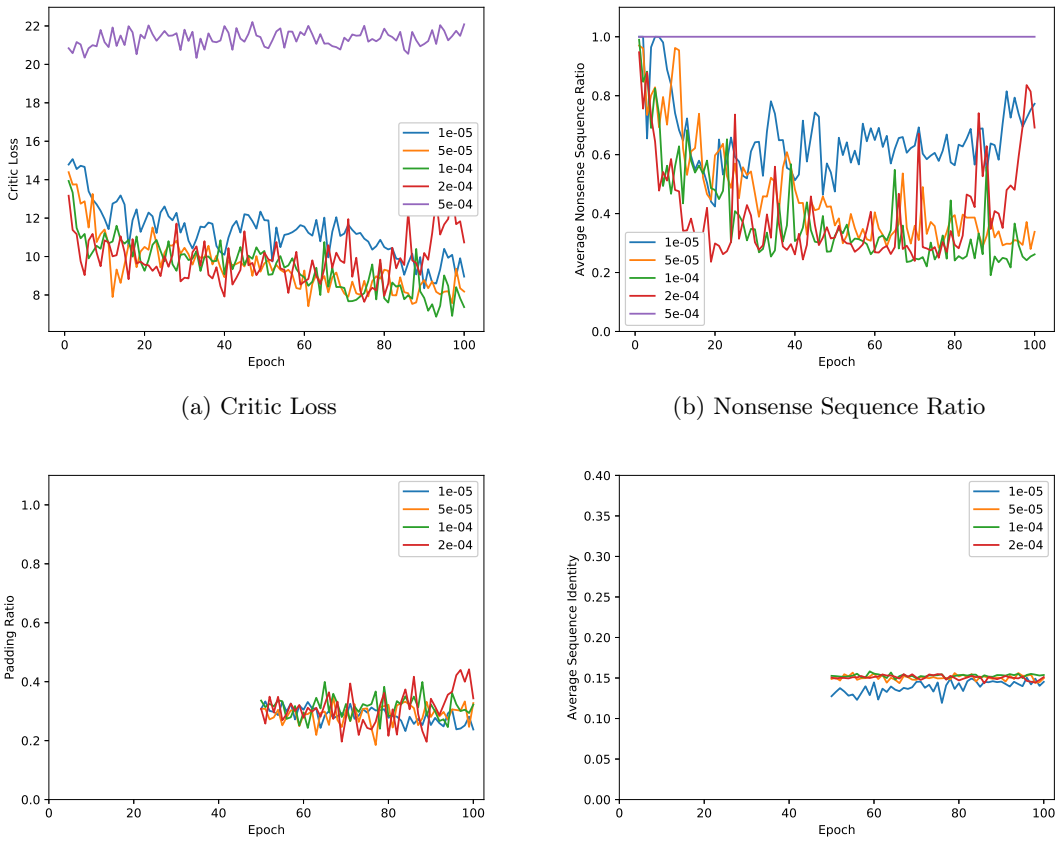
Our final goal is to generate related sequences for some new folds in the future and also some new sequences for the known folds. So we would like the generated sequences are not replicating the training ones, which means we desire the generated sequences to be potentially novel. To measure the sequence novelty, we used the sequence identity as our criteria. The sequence identity is the ratio of the aligned characters in the whole

alignment sequence, and we used the alignment score divided by the smallest alignment length to be the normalized alignment score so that it would be length independent.
For each fold, we generated 100 valid sequences, and for each of the sequences, we calculated the sequence identity between it and all of representative sequence(s) related to the same fold, and then select the maximum ones. Then took the average of these selected scores on all the generated sequences of the fold as the final criteria.

5.2 Hyper-parameters

5.2.1 Initial learning rate

For the training process we used Adam Optimizer to optimize the loss function. Though for Adam Optimizer the learning is not fixed, the initial learning rate would influence it performance. This hyper-parameter is called learning rate in short for the following text. We set it to take values from 0.00001, 0.00005, 0.0001, 0.0002, to 0.0005, and fixed the critic iteration number and noise length to be 10 and 128 respectively.



(a) Critic Loss

(b) Nonsense Sequence Ratio

(c) Padding Ratio

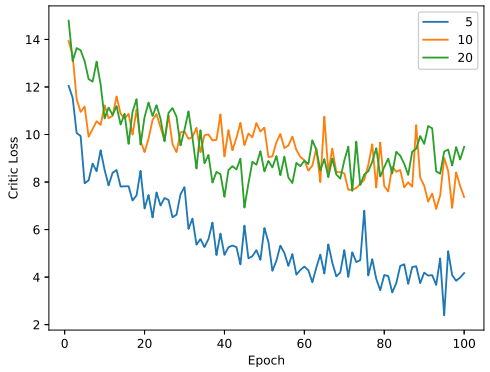
(d) Sequence Identity

Figure S6: Comparison for different initial learning rates when critic iteration number is fixed to 10 and noise length is fixed to 128

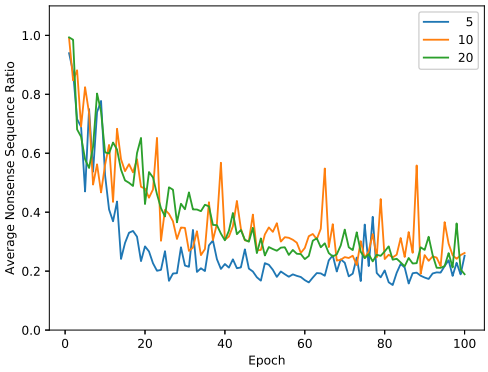
Fig. S6 shows that 1) although the learning rate of 0.0001 had the best critic loss and nonsense ratio, the padding ratio being 1 indicates all-padding sequences; 2) the learning rates 0.0001 and 0.0002 had similar performances and the former has a lower sequence identity. In the end, we choose 0.0001 as the learning rate and fix it for the next rounds of hyper-parameter tuning.

5.2.2 Critic iteration number

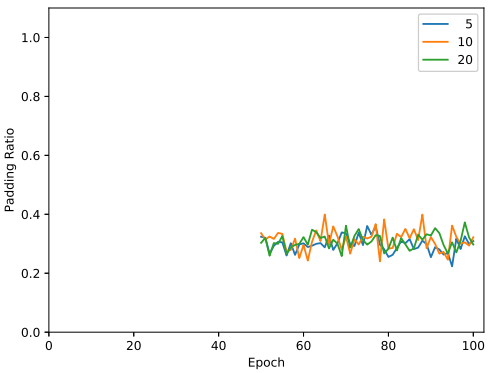
For WGAN there is a generator and a critic, and we train the generator once and the critic several times in each iteration. We fixed the initial learning rate to be 0.0001 and noise length to be 128, and then set the critic iteration number to be 5, 10 and 20 respectively. Based on Fig. S7, we found that when the number is 5 the model performed the best according to the critic loss and nonsense sequence ratio. Moreover, we can see that the sequence identity and padding ratio do not vary much for different critic iteration number. There, we selected 5 as the critic iteration number.
We use around 52 iterations per epoch and 1 batch (size of 64) for the generator. In each of the 52 iterations for the generator, the critic is updated 5 iterations with 5 batches. In total, we trained the whole gcWGAN model for 100 epochs.



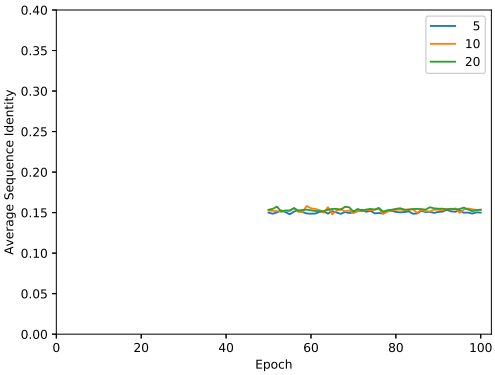
(a) Critic Loss



(b) Nonsense Sequence Ratio



(c) Padding Ratio

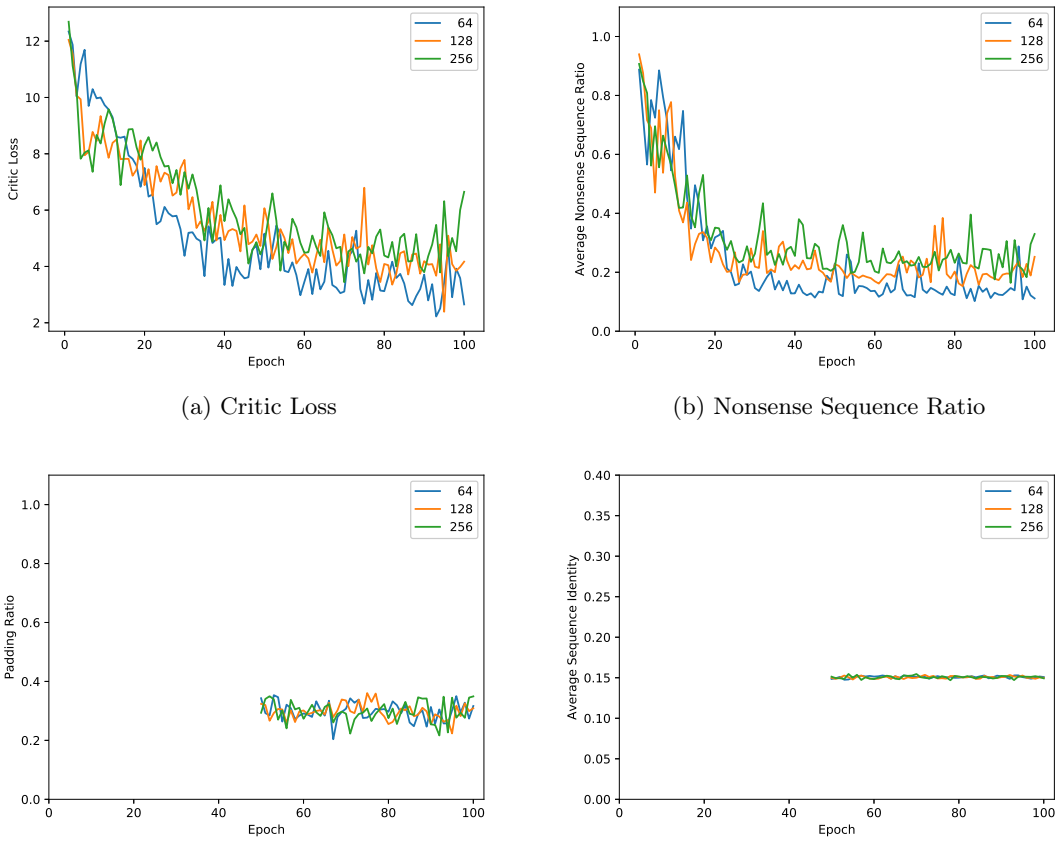


(d) Sequence Identity

Figure S7: Comparison for different numbers of critic iterations when initial learning rate is fixed to 1e-4 and noise length is fixed to 128

5.2.3 Noise length

A random noise vector is an input to the generator and the generator maps the noise distribution to protein sequence distribution. The dimension of the vector is called noise length and also a hyper-parameter. We fixed the learning rate and critic iteration number to be the previous best ones and set the noise length to be 64, 128 and 256 in turn. Like the critic iteration number, we selected 64 as the final noise length according to the critic loss and nonsense sequence ratio based on Fig. S8. Moreover, we can see that the sequence identity and padding ratio do not vary much among the various lengths.



(a) Critic Loss

(b) Nonsense Sequence Ratio

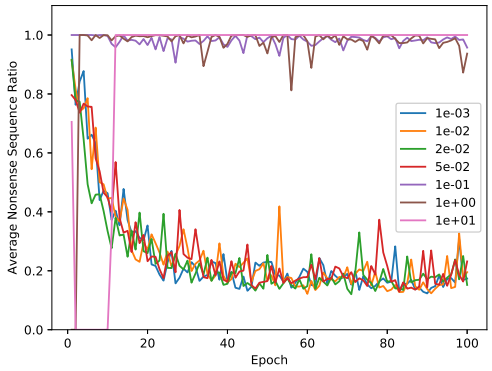
(c) Padding Ratio

(d) Sequence Identity

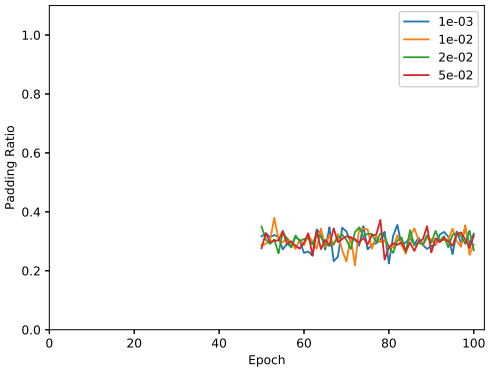
Figure S8: Comparison for different noise length when initial learning rate is fixed to 1e-4 and critic iteration number is fixed to 5

5.2.4 Feedback penalty in gcWGAN (λ_2)

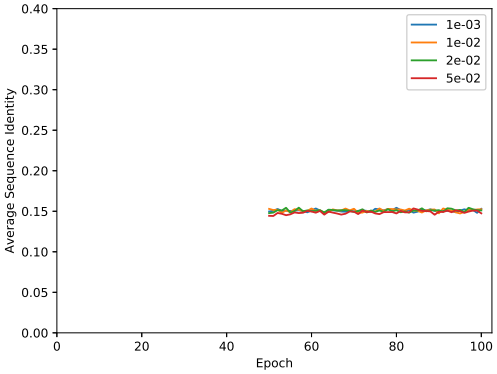
For tuning the hyper-parameter (λ_2) corresponding to the feedback in gcWGAN, we consider the set of values from $\{0.001, 0.01, 0.02, 0.05, 0.1, 1, 10\}$. Since the feedback penalties with different weights are added to the loss function, the overall loss is not comparable, but according to the critic loss we observe that the model will diverge for values larger or equal than 10. For the rest values we refer to the nonsense sequence ratio, padding ratio, sequence identity and an additional sequence instability index that can reflect whether a sequence is protein-like (a value lower than 40 indicates the sequence is stable). From Fig. S9 we find that the nonsense sequence ratio can be nearly 1 if $\lambda_2 \geq 0.1$, and there was no obvious difference for other values according to the four criteria, while 0.02 seems a little better on nonsense sequence ratio.
According to the top-10 accuracy reported by the oracle (which we called yield ratio) for the validation folds, which is shown in Table S5, 0.02 leads to the best performance out of the 4 options. As a result we finally chose 0.02 for λ_2.



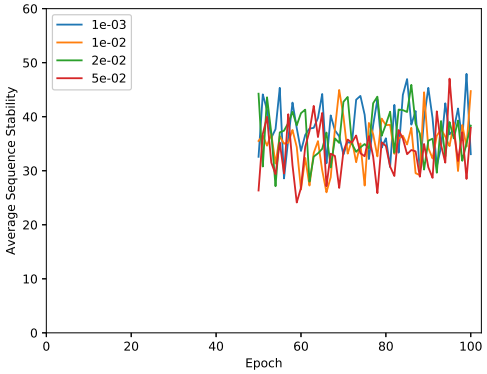
(a) Nonsense Sequence Ratio



(b) Padding Ratio



(c) Sequence Identity



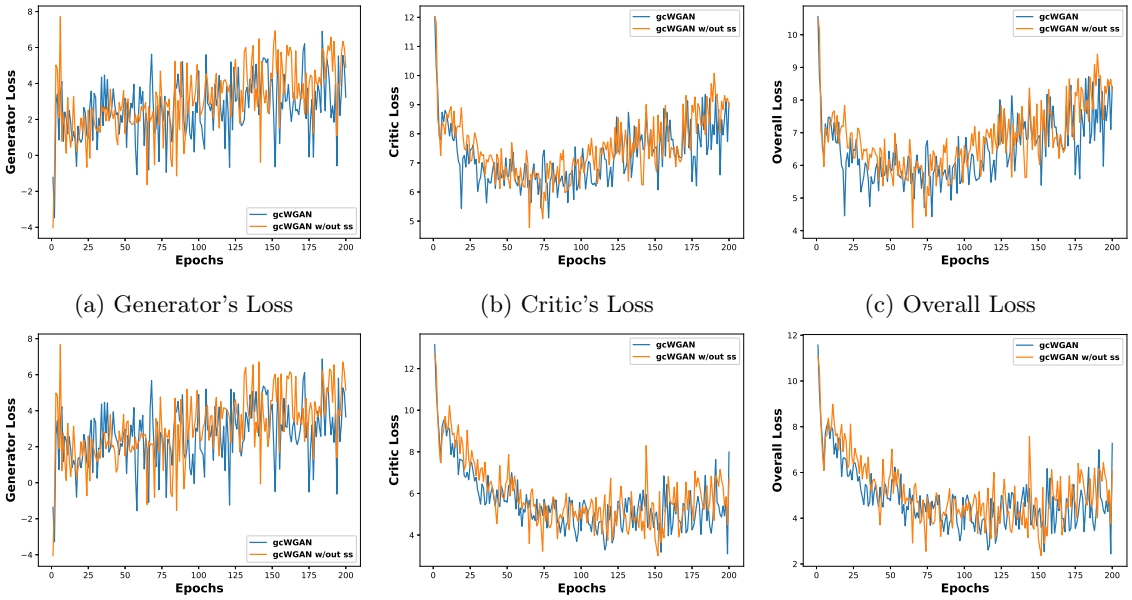
(d) Sequence Stability

Figure S9: Comparison for different feedback penalty weight when other hyper-parameters are fixed to be the ones got from cWGAN.

Feedback penalty (λ_2)	0.001	0.01	0.02	0.05
all	5.4e-4	5.8e-4	9.6e-4	4.4e-4
Fold Class Breakdowns				
a	4.2e-5	1.5e-4	3.9e-4	8.7e-5
b	1.5e-3	6.3e-4	1.6e-3	1.1e-3
c	1.7e-3	2.7e-3	4.5e-3	2.6e-3
d	4.7e-4	5.7e-4	7.4e-4	1.7e-4
e	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5
f	8.7e-4	1.2e-3	1.1e-3	4.2e-4
g	$<$ 1e-5	$<$ 1e-5	8.8e-5	$<$ 1e-5
Sequence-Availability Class Breakdowns				
easy	2.7e-3	4.2e-3	6.9e-3	1.1e-3
medium	2.2e-3	1.3e-3	1.1e-3	1.1e-3
hard	2.3e-4	2.7e-4	5.7e-4	3.3e-4

Table S5: The yield ratios on the validation set for different feedback penalty weight.

6 Effect of Semi-supervision on gcWGAN



(a) Generator's Loss (b) Critic's Loss (c) Overall Loss

(d) Generator's Loss (e) Critic's Loss (f) Overall Loss

Figure S10: Comparison of gcWGAN training when initialized using semi-supervised cWGAN (3-step training) or using cWGAN (without unsupervised pre-training) for generator, critic and overall losses. The top panel is for the training set and the bottom panel is for the validation set.

	One-sided paired t-test		One-sided paired Wilcoxon signed-rank test	
Epochs	statistic	p-value	statistic	p-value
81-100	-1.69	0.05	58	0.04
81-200	-2.94	0.0019	2555	0.0024
1-100	-3.67	0.0001	1476	0.0001
101-200	-2.56	0.0058	1827	0.0082
181-200	-2.29	0.0165	51	0.0220
1-200	-4.25	1.58e-5	6655	1.72e-5

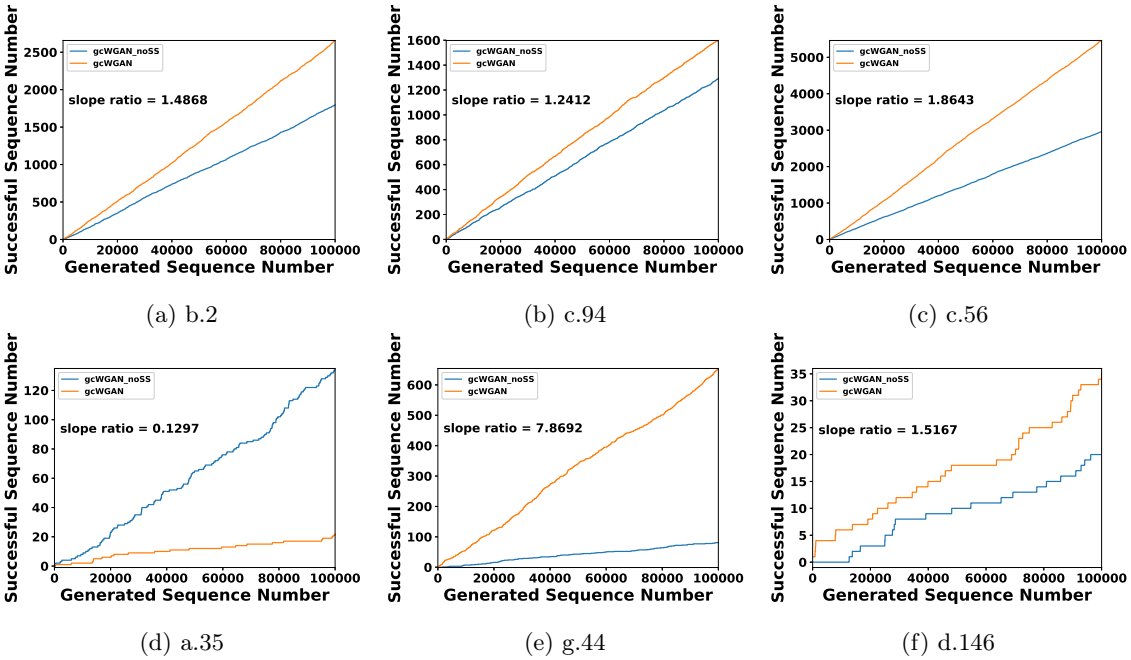
Table S6: Statistical tests on whether the overall losses with semi-supervision are lower than those without.

We first formally define various losses below:

$$\begin{aligned} L_{\text{Oracle}} &= -\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[I_{\text{target}}(\tilde{\boldsymbol{x}})\log O(\tilde{\boldsymbol{x}}) + (1 - I_{\text{target}}(\tilde{\boldsymbol{x}}))\log(1 - O(\tilde{\boldsymbol{x}}))] \\ L_{\text{Gen}} &= -\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}}|\boldsymbol{y})] + \lambda_2 L_{\text{Oracle}} \\ L_{\text{Critic}} &= \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x}|\boldsymbol{y})] - \mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}}|\boldsymbol{y})] - \lambda_1\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}[(||\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}}|\boldsymbol{y})||_2 - 1)^2] \\ L_{\text{Overall}} &= L_{\text{Critic}} + \lambda_2 L_{\text{Oracle}} \end{aligned} \tag{4}$$

To check the impact of semi-supervision on the convergence and stability of model training, we trained both gcWGAN models (with and without semi-supervised strategy) for 200 epochs and observed the overall loss and its decomposition (generator's and critics loss) for both training and validation sets (Figure S10). Toward our goal to minimize the overall loss, the semi-supervised training had lower values. Specifically, we performed one-sided paired t-test and Wilcoxon signed-rank test for the validation set and the resulting P values confirmed the observed trends.

For further quantitative assessment, we assessed the sequences generated from the two training strategies, based on yield ratio. In details, comparing these two training strategies in larger scale protein sequence generation (100,000 sequences per fold) for the 6 case-study folds, we observed that 5 out of 6 folds had higher yield ratio in gcWGAN with semi-supervised initialization rather than just initializing based on supervised (shown in figure S11).



(a) b.2 (b) c.94 (c) c.56

(d) a.35 (e) g.44 (f) d.146

Figure S11: The number of "yields" (successful sequences according to the oracle, a fold predictor) versus the number of sequences generated by gcWGAN with or without semi-supervision.

7 Assessing generated sequences prior to the final oracle check

Once the generator is trained and chosen, we first generated up to 10^5 "valid" sequences (passing the nonsense check) for each fold. Specifically, starting with 10^3 valid sequences per fold, we incrementally generate sequences at a stepsize around 10^3 (batch size being 1,400 followed by removing nonsense sequences), until finding at least 10 yields or generating 10^5 sequences, whichever arrives earlier.
For these sequences generated for each fold, we examined their yield ratios (the accuracy according to the oracle's top-10 prediction). This is an intermediate sanity check as the final sequences will be only those that pass the oracle's check. We estimate "yield ratios" (success rates) for generated valid sequences (passing the nonsense check) using the oracle (fold predictor). Specifically, a sequence is declared a "yield" if its top-10 oracle-predicted folds include the target; and the yield ratio is the portion of those yields.

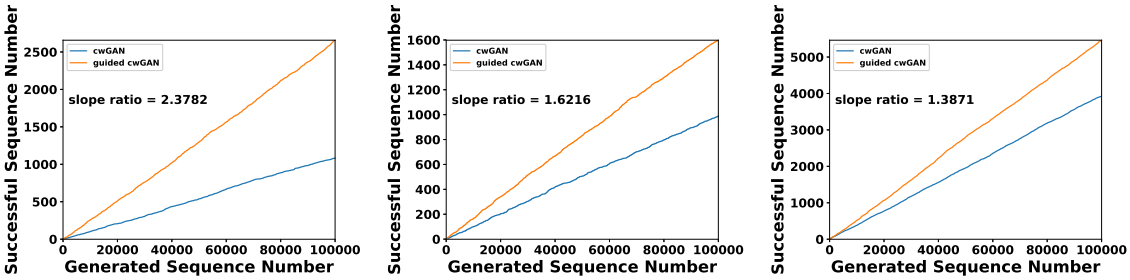
7.1 gcWGAN with oracle feedback improves yields

Yield Ratio	cVAE			cWGAN			guided-cWGAN		
	training	validation	test	training	validation	test	training	validation	test
all	2.1e-2	1.9e-3	7.5e-3	1.3e-2	6.7e-4	1.4e-3	1.3e-2	9.6e-4	2.5e-3
Fold Class Breakdowns									
a	6.7e-3	4.5e-4	$<$ 1e-5	1.7e-3	9.4e-5	7.9e-5	2.4e-3	3.9e-4	2.8e-4
b	5.5e-2	1.2e-2	5.3e-2	2.6e-2	1.4e-3	3.5e-3	2.5e-2	1.6e-3	7.0e-3
c	6.9e-2	1.1e-3	2.2e-5	4.9e-2	3.3e-3	8.6e-3	5.3e-2	4.5e-3	1.2e-2
d	6.2e-4	$<$ 1e-5	1.4e-5	4.1e-3	4.5e-4	7.5e-4	5.0e-3	7.4e-4	1.1e-3
e	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	2.3e-4	$<$ 1e-5	$<$ 1e-5	2.1e-4	$<$ 1e-5	$<$ 1e-5
f	9.4e-2	4.1e-5	$<$ 1e-5	5.2e-2	1.5e-3	2.2e-5	3.2e-2	1.1e-3	$<$ 1e-5
g	7.7e-4	$<$ 1e-5	1.1e-3	2.5e-4	1.0e-5	9.1e-5	1.9e-3	8.8e-5	9.4e-4
Sequence-Availability Class Breakdowns									
easy	5.6e-2	2.3e-3	2.2e-3	3.3e-2	3.0e-3	1.3e-2	3.6e-2	6.9e-3	2.1e-2
medium	2.3e-2	1.8e-2	3.6e-2	1.3e-2	1.3e-3	8.1e-4	1.2e-2	1.1e-3	1.4e-3
hard	1.6e-3	9.4e-5	$<$ 1e-5	1.8e-3	4.7e-4	4.6e-4	1.9e-3	5.7e-4	8.9e-4
Oracle-Accuracy Class Breakdowns									
$0 \sim 0.25$	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	1.3e-5	$<$ 1e-5	$<$ 1e-5	2.1e-5	$<$ 1e-5	$<$ 1e-5
$0.25 \sim 0.5$	3.6e-3	$<$ 1e-5	$<$ 1e-5	1.1e-3	1.8e-5	2.6e-5	9.5e-4	2.7e-4	2.1e-5
$0.5 \sim 0.75$	1.1e-5	$<$ 1e-5	$<$ 1e-5	2.0e-3	1.4e-3	1.2e-3	3.6e-3	3.1e-3	2.0e-3
$0.75 \sim 1$	3.9e-2	4.1e-3	1.5e-2	2.3e-2	1.2e-3	2.5e-3	2.3e-2	1.5e-3	4.3e-3

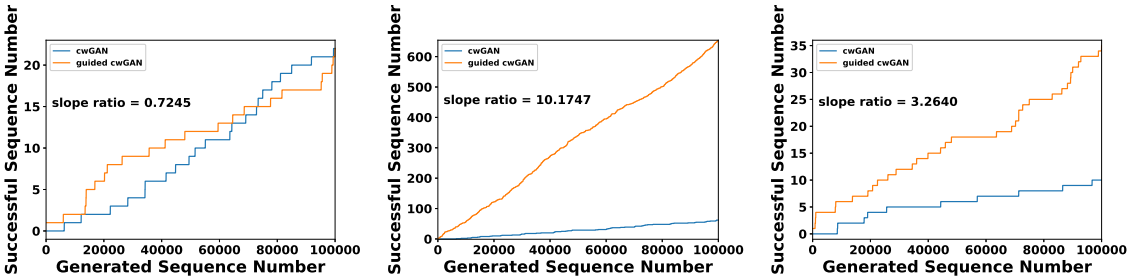
Table S7: Comparing yield ratios for cVAE, cWGAN and gcWGAN for our training, validation, and test sets.

yield ratio	cVAE			cWGAN			guided-cWGAN		
	training*	validation*	test*	training*	validation*	test*	training*	validation*	test*
all	3.7e-2	2.2e-3	1.5e-4	1.8e-2	5.4e-4	1.0e-3	1.8e-2	8.7e-4	1.5e-3
Fold Class Breakdowns									
a	3.7e-2	4.6e-4	$<$ 1e-5	8.1e-4	9.7e-5	6.7e-5	2.2e-3	4.0e-4	2.9e-4
b	9.8e-2	2.0e-2	$<$ 1e-5	5.0e-2	1.9e-3	2.6e-4	4.8e-2	1.9e-3	1.2e-4
c	4.7e-2	7.0e-4	2.5e-5	1.9e-2	1.6e-3	9.8e-3	2.2e-2	4.4e-3	1.4e-2
d	5.8e-4	$<$ 1e-5	1.9e-5	3.0e-3	5.2e-4	4.7e-4	4.3e-3	8.8e-4	4.2e-4
e	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	1.9e-5	$<$ 1e-5	$<$ 1e-5
f	1.6e-1	4.1e-5	$<$ 1e-5	7.5e-2	1.5e-3	$<$ 1e-5	4.5e-2	1.1e-3	$<$ 1e-5
g	2.5e-3	$<$ 1e-5	1.3e-3	2.9e-4	1.0e-5	6.0e-5	3.1e-3	8.8e-5	3.8e-4
Sequence-Availability Class Breakdowns									
easy	8.3e-2	3.3e-3	4.2e-3	3.7e-2	3.6e-3	1.2e-2	3.9e-2	9.0e-3	1.0e-2
medium	2.7e-2	2.3e-2	$<$ 1e-5	1.4e-2	1.4e-3	1.1e-3	1.3e-2	1.2e-3	1.6e-3
hard	$<$ 1e-5	3.7e-5	$<$ 1e-5	2.9e-4	3.0e-4	5.3e-4	2.9e-4	4.4e-4	1.0e-3
Oracle-Accuracy Class Breakdowns									
$0 \sim 0.25$	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	4.3e-5	$<$ 1e-5	$<$ 1e-5	8.0e-5	$<$ 1e-5	$<$ 1e-5
$0.25 \sim 0.5$	7.1e-3	$<$ 1e-5	$<$ 1e-5	1.0e-3	1.8e-5	5.2e-5	1.3e-3	2.7e-4	4.1e-5
$0.5 \sim 0.75$	$<$ 1e-5	$<$ 1e-5	$<$ 1e-5	2.8e-3	1.4e-3	3.9e-4	3.6e-3	3.1e-3	6.1e-4
$0.75 \sim 1$	6.3e-2	5.3e-3	2.8e-4	2.9e-2	9.3e-4	1.8e-3	2.9e-2	1.3e-3	2.6e-3

Table S8: Comparing yield ratios for cVAE, cWGAN and gcWGAN over (*) shared training, validation, and test sets. Retraining cVAE was infeasible due to that training scripts were not accessible.



(a) b.2 (b) c.94 (c) c.56



(d) a.35 (e) g.44 (f) d.146

Figure S12: The number of "yields" versus the number of sequences generated by cWGAN/gcWGAN for selected folds.

7.2 cWGAN and gcWGAN have higher yields for more folds than cVAE

	cVAE		cWGAN		gcWGAN	
	yield ratio	fold coverage	yield ratio	fold coverage	yield ratio	fold coverage
all	7.5e-3	0.084	1.4e-3	0.327	2.5e-3	0.290
Fold Class Breakdowns						
a	$<$ 1e-5	0.000	7.9e-5	0.286	2.8e-4	0.250
b	5.3e-2	0.200	3.5e-3	0.467	7.0e-3	0.267
c	2.2e-5	0.125	8.6e-3	0.750	1.2e-2	0.750
d	1.4e-5	0.049	7.5e-4	0.268	1.1e-3	0.195
e	$<$ 1e-5	0.000	$<$ 1e-5	0.000	$<$ 1e-5	1.000
f	$<$ 1e-5	0.333	2.2e-5	0.333	$<$ 1e-5	0.000
g	1.1e-3	0.182	9.1e-5	0.182	9.4e-4	0.455
Sequence-Availability Class Breakdowns						
easy	2.2e-3	0.625	1.3e-2	1.000	2.1e-2	0.875
medium	3.6e-2	0.136	8.1e-4	0.500	1.4e-3	0.364
hard	$<$ 1e-5	0.013	4.6e-4	0.208	8.9e-4	0.208
Oracle-Accuracy Class Breakdowns						
$0 \sim 0.25$	$<$ 1e-5	0.032	$<$ 1e-5	0.097	$<$ 1e-5	0.032
$0.25 \sim 0.5$	$<$ 1e-5	0.000	2.6e-5	0.250	2.1e-5	0.250
$0.5 \sim 0.75$	$<$ 1e-5	0.053	1.2e-3	0.368	2.0e-3	0.368
$0.75 \sim 1$	1.5e-2	0.132	2.5e-3	0.453	4.3e-3	0.415

Table S9: Yield comparison among cVAE, cWGAN and guided-cWGAN for the test set.

	cVAE		cWGAN		gcWGAN	
	yield ratio	fold coverage	yield ratio	fold coverage	yield ratio	fold coverage
all	1.5e-4	0.048	1.0e-3	0.321	1.5e-3	0.298
Fold Class Breakdowns						
a	$<$ 1e-5	0.000	6.7e-5	0.280	2.9e-4	0.240
b	$<$ 1e-5	0.000	2.6e-4	0.500	1.2e-4	0.200
c	2.5e-5	0.143	9.8e-3	0.714	1.4e-2	0.714
d	1.9e-5	0.067	4.7e-4	0.300	4.2e-4	0.233
e	$<$ 1e-5	0.000	$<$ 1e-5	0.000	$<$ 1e-5	1.000
f	$<$ 1e-5	0.000	$<$ 1e-5	0.000	$<$ 1e-5	0.000
g	1.3e-3	0.111	6.0e-5	0.111	3.8e-4	0.444
Sequence-Availability Class Breakdowns						
easy	4.2e-3	0.667	1.2e-2	1.000	1.0e-2	1.000
medium	$<$ 1e-5	0.067	1.1e-3	0.600	1.6e-3	0.400
hard	$<$ 1e-5	0.015	5.3e-4	0.227	1.0e-3	0.242
Oracle-Accuracy Class Breakdowns						
$0 \sim 0.25$	$<$ 1e-5	0.000	$<$ 1e-5	0.08	$<$ 1e-5	0.04
$0.25 \sim 0.5$	$<$ 1e-5	0.000	5.2e-5	0.5	4.1e-5	0.5
$0.5 \sim 0.75$	$<$ 1e-5	0.083	3.9e-4	0.333	6.1e-4	0.333
$0.75 \sim 1$	2.8e-4	0.067	1.8e-3	0.444	2.6e-3	0.422

Table S10: Yield comparison among cVAE, cWGAN and guided-cWGAN for the test set excluding the cVAE training folds (test* or common test set). Retraining cVAE was not possible as training scripts were not released.

Order	cVAE	cWGAN	gcWGAN
1	b.9 (7.9e-1)	b.1 (3.8e-2)	b.1 (8.6e-2)
2	g.39 (1.2e-2)	c.56 (3.1e-2)	c.56 (6.0e-2)
3	b.1 (5.0e-3)	c.52 (2.2e-2)	d.79 (3.1e-2)
4	d.92 (5.6e-4)	d.79 (1.6e-2)	b.2 (1.8e-2)
5	c.56 (1.7e-4)	d.92 (1.3e-2)	c.52 (1.8e-2)

Table S11: Top 5 folds with the highest yield ratio for cVAE, cWGAN and guided-cWGAN

Cases	cVAE	cWGAN	gcWGAN
b.2	7e-5	1.2e-2	1.8e-2
c.94	$<$ 1e-5	1.1e-2	1.0e-2
c.56	1.7e-4	3.1e-2	6.0e-2
a.35	$<$ 1e-5	5.3e-4	5.6e-4
g.44	1.0e-5	4.6e-4	6.9e-3
d.146	$<$ 1e-5	4.4e-5	3.0e-4

Table S12: Yield ratio for 6 selected representative folds for cVAE, cWGAN and guided-cWGAN

In order to find out whether there can be an intersection between the test set of gcWGAN and the training set of cVAE, for each fold in our test set we calculated the symmetric TM-score between it and the cVAE training data, and then selected the PDB with the highest score to show the closeness between the two dataset. Table S13 shows the the results for the 6 selected cases.

Cases	PDB info	best TM-score	related PDB in cVAE training set
b.2	2AXW (chain A and resi 1-121)	0.55601	2CWR (chain A)
c.94	4F19 (chain A and resi 1001-1370)	0.36847	2A9Q (chain A)
c.56	1RTQ (chain A)	0.47206	2LV8 (chain A)
a.35	2XI8 (chain A)	0.83240	2LYP (chain A)
g.44	2BAY (chain A)	0.74036	3L1X (chain A)
d.146	1UXY (chain A and resi 101-343)	0.45715	1KP6 (chain A)

Table S13: The TM-scores between the 6 selected representative folds and their closest structures in the training set of cVAE.

7.3 Sequence identity results for generated designs and natural sequences

We also compare sequence identity between generated and natural sequences. For each fold, a trained model (cVAE, cWGAN or gcWGAN) generates 100 sequences, each of which is pairwise-aligned to all natural sequences of the fold and scored by the maximum sequence identity. We provide the average sequence identities over shared training, validation, and test sets (including their breakdowns) below. We notice that the three models had similar test-set sequence identities and cVAE with higher training-set sequence identities showed overfit.

Identity	cVAE			cWGAN			guided-cWGAN		
	training*	validation*	test*	training*	validation*	test*	training*	validation*	test*
all	0.323	0.156	0.160	0.187	0.153	0.155	0.186	0.152	0.154
Fold Class Breakdowns									
a	0.321	0.157	0.160	0.188	0.154	0.157	0.186	0.153	0.153
b	0.332	0.159	0.156	0.195	0.159	0.154	0.193	0.158	0.153
c	0.292	0.171	0.176	0.191	0.161	0.174	0.192	0.161	0.175
d	0.336	0.159	0.159	0.184	0.158	0.155	0.183	0.157	0.154
e	0.193	0.141	0.147	0.169	0.150	0.140	0.179	0.152	0.149
f	0.175	0.131	0.153	0.168	0.139	0.157	0.163	0.142	0.160
g	0.356	0.149	0.155	0.179	0.136	0.140	0.177	0.135	0.140
Sequence-Availability Class Breakdowns									
easy	0.409	0.213	0.213	0.209	0.203	0.198	0.207	0.202	0.197
medium	0.310	0.165	0.190	0.186	0.177	0.182	0.184	0.176	0.182
hard	0.239	0.152	0.150	0.163	0.148	0.147	0.162	0.148	0.146

Table S14: Comparing cVAE, cWGAN and gcWGAN designs in their average (max) sequence identities to natural sequences in the (*) shared training, validation, and test sets.

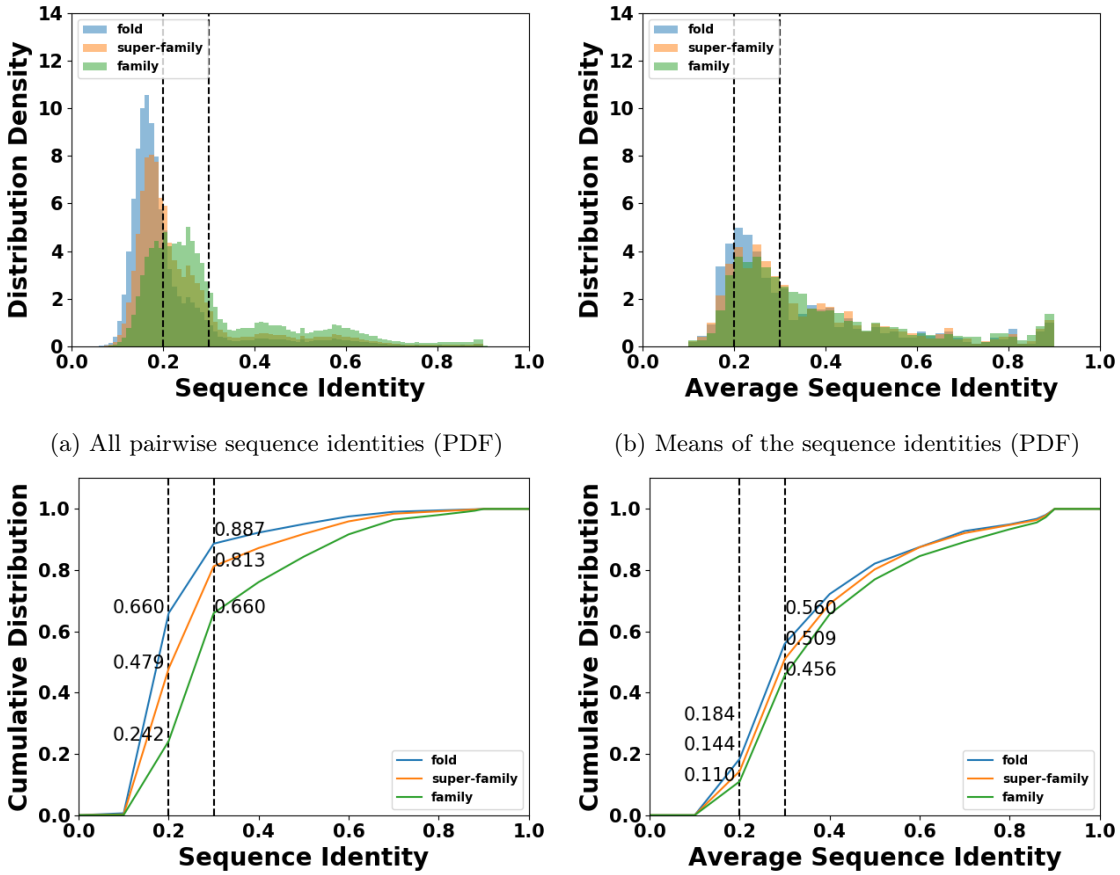
To interpret the sequence identity results above in a right context, it is important to examine the sequence identities among natural sequences.

We first examine sequence identities among all pairs (90% redundancy level) of natural sequences belonging to the same fold, superfamily, or family and plot their probability density function (PDF) and cumulative distribution function (CDF) in S13a and S13c, respectively. We note that 66%, 48%, and 24% of sequence pairs within the same fold, superfamily, and family respectively are of sequence identity within 20%. Even for sequences in the same family which are prone to share close homology, a nontrivial fraction of pairs have sequence identities below 0.2. Examples include SCOPe ID d1f7ua3 and d1iq0a3 that belong to the same family of d.67.2.1 but have a sequence identity of just 0.148.

In the case of folds, no sequence similarity is implied by definition. Unlike superfamily or family where sequences share distant or close homology, the SCOPe hierarchy of fold by definition does not imply common evolutionary origin. Indeed, two-thirds of sequence pairs within a fold are very dissimilar. This is the data distribution our models learned from the training sequences.

We also calculated average sequence identity of each fold and report their PDFs and CDFs in S13b and S13d,

respectively. Still over 18% and 56% folds have average sequence identity within 20%, and 30% respectively. This would be the data distribution our models learn from the training sequences if fold-level re-weighting were introduced.
Therefore, the data distribution in natural sequences shows that sequences in the same fold are only similar to each other at a low level. The mode and the average are both below 0.20, which is consistent with the level observed for the model-generated sequences (Table S14).



(a) All pairwise sequence identities (PDF)

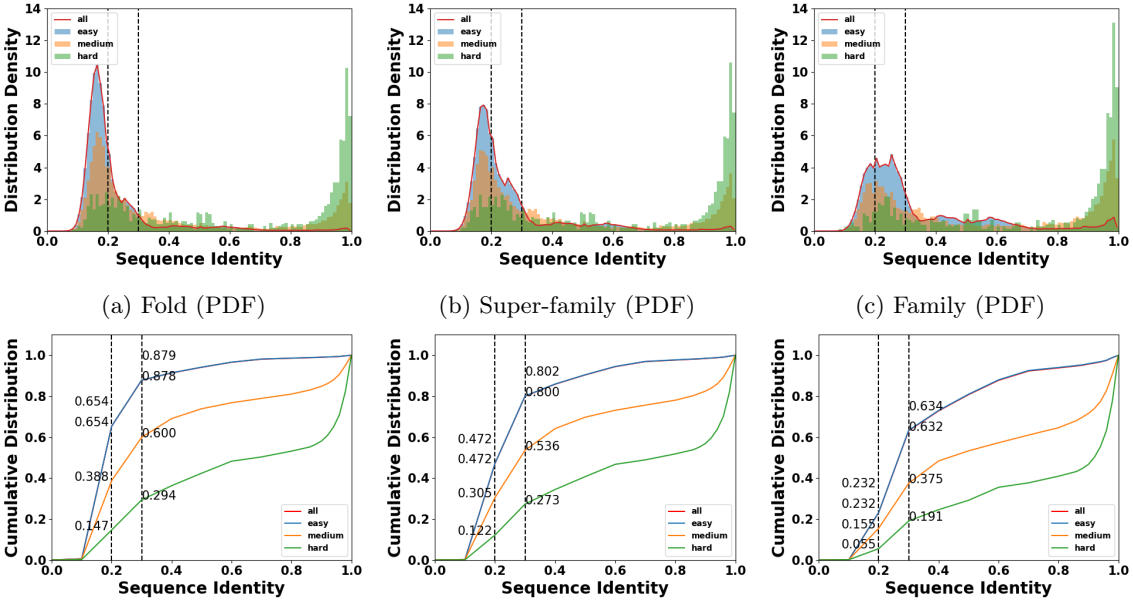
(b) Means of the sequence identities (PDF)

(c) All pairwise sequence identities (CDF)

(d) Means of the sequence identities (CDF)

Figure S13: Distributions of (a,c) sequence identities or (b,d) their average for sequence pairs of the same fold, superfamily, or family. 90% sequence redundancy level is considered here.

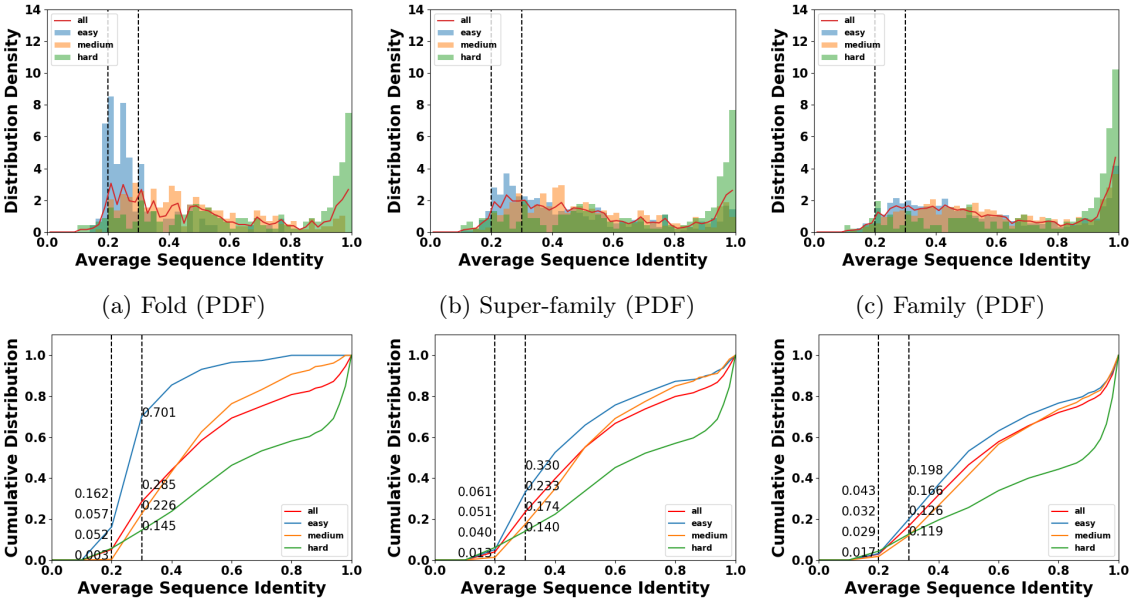
For completeness, we additionally provide the PDF and CDF of (average) sequence identities within a fold, superfamily, and family when sequence redundancy cutoff is at 100% (see Fig. S14). Furthermore, we provide the breakdowns of those PDFs and CDFs for various subsets of folds (easy, medium, and difficult) in Fig. S15.



(a) Fold (PDF) (b) Super-family (PDF) (c) Family (PDF)

(d) Fold (CDF) (e) Super-family (CDF) (f) Family (CDF)

Figure S14: Distributions of sequence identities for sequence pairs of the same fold, superfamily, or family (including breakdowns according to sequence abundancy). 100% sequence redundancy level is considered here.



(a) Fold (PDF) (b) Super-family (PDF) (c) Family (PDF)

(d) Fold (CDF) (e) Super-family (CDF) (f) Family (CDF)

Figure S15: Distributions of average sequence identities of folds, superfamilies, or families (including their breakdowns according to sequence abundancy). 100% sequence redundancy level is considered here.

8 cVAE pipeline

We adopted the following pipeline for applying cVAE (Greener et al., 2018):

- we first download the software from https://github.com/psipred/protein-vae. In the GitHub repository, the topology files are provided to assign a grammar to each PDB in their dataset: *topology_data/assign_by_tmscores.txt* and *topology_data/assign_directly.txt*.

- In order to assign the grammar to each given fold in our test set, we calculated the TM-scores between the representative PDB of that fold and all PDBs in the cVAE topology files whose grammars were not assigned on TM-scores (the PDBs in *topology_data/assign_directly.txt* and the corresponding PDBs in *topology_data/assign_by_tmscores.txt*, which are the assigned representative SCOP proteins). And we choose the cVAE PDB with the highest symmetric TM-score and used its grammar for the test fold. Among all 107 test folds, we find 25 folds with best TM-score larger than 0.5 and 84 folds with best TM-score larger than 0.4. Specifically, for 7 case-study folds, we show their best TM-scores in Table S15. All but only one fold had TM-scores above or close to 0.5 and thus had decent-quality grammars assigned. Meanwhile, we note from Table S13 that 3 of the test folds were in cVAE's training set, judging from the TM-scores being above 0.5. (The information on cVAE training set had not been accessible when our test set was chosen.)

Fold	a.35	b.2	c.56	c.94	d.146	g.44	novel
Best TM-score	1.0	0.55	0.46	0.36	0.46	0.69	0.42

Table S15: Best TM-score for 7 case-study folds.

- We then put the assigned grammar into the file 'gram2seq_example.txt' and then run

  ```
  python gram_to_seq.py -infile gram2seq.txt -numout n
  ```

  to generate n sequences for that grammar, where n is a user-defined constant.

9 Ab initio protein structure prediction (Rosetta) pipeline

Followings is the pipeline for abinitio protein structure prediction through Rosetta software as suggested in their tutorial and Das et al. (2009):

Creating fragment library: At first, we predict the secondary structures based on sequence information only through PSIPRED software McGuffin et al. (2000). Then we create 3-mer and 9-mer fragment library with the following command from Rosetta:

```
/rosetta_bin_linux_2018.33.60351_bundle/main/source/bin/fragment_picker.linuxgccrelease \
        -in::file::vall ../../tools/fragment_tools/vall.jul19.2011 \
        -in:file:fasta ${fasta_file} \
        -frags:scoring:config ${score_file} \
        -frags:ss_pred ${ss_file} psipred \
        -frags:frag_sizes 3 9 \
        -out::file::frag_prefix ${output_name}
```

AbinitioRelax protein structure prediction: Then, we run 10,000 trajectories (100 parallel jobs in which 100 pdbs is predicted for each job)for each sequence with the following command from Rosetta:

```
/rosetta_bin_linux_2018.33.60351_bundle/main/source/bin/AbinitioRelax.linuxgccrelease \
        -in:file:fasta ${seq} \
        -in:file:frag3 ${3_mers} \
        -in:file:frag9 ${9_mers} \
        -abinitio:relax \
```

```
-abinitio::increase_cycles 10 \
-abinitio::rg_reweight 0.5 \
-abinitio::rsd_wt_helix 0.5 \
-abinitio::rsd_wt_loop 0.5 \
-out:pdb \
-out:file:silent ${output_name}_silent.out \
-constant_seed \
-jran ${job_id} \
-seed_offset ${job_off} \
-kill_hairpins ${ss_file} \
-relax::fast \
-use_filters true \
-psipred_ss2 ${ss_file} \
-out:path ${output_path} \
-nstruct 100
```

Combining results from abinitioRelax trajectories: Then we combine all the silent output files of 100 parallel job with the following command:

```
/rosetta_bin_linux_2018.33.60351_bundle/main/source/bin/combine_silent.linuxgccrelease
```

Rosetta Clustering: Then, we perform Rosetta default clustering with the following command:

```
../../../rosetta_bin_linux_2018.33.60351_bundle/main/source/bin/cluster.linuxgccrelease \
-database ../../../rosetta_bin_linux_2018.33.60351_bundle/main/database \
-in:file:silent final_silent.out \
-cluster:input_score_filter ${filter} \
-cluster:radius ${radius} \
-out:file:silent ${output_file}_cluster.out \
-cluster:limit_clusters 10 \
-cluster:sort_groups_by_energy
```

Extracting representative pdbs: Then we select the cluster representatives (lowest energy). Since we set the maximum number of clusters to be 10, therefore we will have at most 10 representative predicted structure for each sequence:

```
../../../rosetta_bin_linux_2018.33.60351_bundle/main/source/bin/score.linuxgccrelease \
-database ../../../rosetta_bin_linux_2018.33.60351_bundle/main/database \
-in:file:silent ${silent_file}_cluster.out \
-out:output
```

10 Rosetta de novo design pipeline

RosettaScripts (Fleishman et al., 2011), common xml-scriptable interface, has been utilized to design sequences for novel fold. We have followed the similar procedure as recommended by Rosetta developers (Koga et al., 2012). To show that gcWGAN can improve quantity and quality of designed sequences in Rosetta design, we have consider 10 different set f designs. In 5 of these set of designs, we utilized the recommended initial starting wild-type sequence by Rosetta developers (Koga et al., 2012). In the other 5, we calculate the maximum probable amino acids from the marginal distributions at each position and consider these amino acids as the initial starting wild-type sequence. To calculate the marginal distributions at each position for novel fold, we generated 10,000 sequences using gcWGAN. Then at each position we calculate the marginal distributions and sort them from most probable ones to least probable ones. To reduce the search space for Rosetta design, we consider 5 different thresholds (1, 0.99, 0.95, 0.9 and 0.5) by which we

will truncate the amino acids whose CDF exceed the threshold. For each set of design, we run 20 4-day parallel jobs to design sequences for novel fold. The Rosseta script written for this task is shown in the following:

```
<ROSETTASCRIPTS>

        <SCOREFXNS>
           <ScoreFunction name="SFXN1" weights="score3" >
                <Reweight scoretype="hbond_sr_bb" weight="1.0" />
                <Reweight scoretype="hbond_lr_bb" weight="1.0" />
           </ScoreFunction>
           <ScoreFunction name="SFXN2" weights="score3" >
                <Reweight scoretype="hbond_sr_bb" weight="1.0" />
                <Reweight scoretype="hbond_lr_bb" weight="1.0" />
           </ScoreFunction>
           <ScoreFunction name="SFXN3" weights="myweight" >
                <Reweight scoretype="atom_pair_constraint" weight="1.0" />
           </ScoreFunction>
        </SCOREFXNS>
        <FILTERS>

           <HelixPairing  name="hp1" helix_pairings="1-2.A" dist="13.0" cross="30.0"
           align="5.0" blueprint="./input1.bbskel" />
           <HelixKink          name="hk1" blueprint="./input1.bbskel" />
           <CompoundStatement name="cm1" >
                <AND filter_name="hp1"/>
                <AND filter_name="hk1"/>
           </CompoundStatement>
           <HelixKink     name="hk2" blueprint="./input2.bbskel" />
           <HelixPairing  name="hp2" helix_pairings="1-2.A;3-4.A" dist="13.0" cross="30.0"
           align="5.0" blueprint="./input2.bbskel" />
           <CompoundStatement name="cm2" >
                <AND filter_name="hk2"/>
                <AND filter_name="hp2"/>
           </CompoundStatement>
           <SecondaryStructure name="ss" blueprint="./input2.bbskel" />
           <ScoreType name="sc" scorefxn="REF2015" score_type="total_score" threshold="-170"/>
           <PackStat name="pstat" threshold="0.6" />


        </FILTERS>
        <TASKOPERATIONS>

           <LimitAromaChi2 name="limitchi2" />
           <LayerDesign name="layer_all"           layer="core_boundary_surface"
           core_H="15" core_E="15" core_L="25" surface_H="60" surface_E="60"
           surface_L="40" pore_radius="2.0" verbose="true" />

        </TASKOPERATIONS>
        <MOVERS>
           <SetSecStructEnergies name="sse1" scorefxn="SFXN1" blueprint="./input1.bbskel" />
           <BluePrintBDR name="bdr1" scorefxn="SFXN1" use_abego_bias="1"
           blueprint="./input1.bbskel" />
           <LoopOver     name="lover1" mover_name="bdr1" filter_name="cm1"
           iterations="1000" drift="0" ms_whenfail="FAIL_DO_NOT_RETRY"/>
```

```
        <SetSecStructEnergies name="sse2" scorefxn="SFXN2"
        blueprint="./input2.bbskel" />
        <BluePrintBDR name="bdr2" scorefxn="SFXN2" use_abego_bias="1"
        blueprint="./input2.bbskel" />
        <LoopOver     name="lover2" mover_name="bdr2" filter_name="cm2"
        iterations="1000" drift="0" ms_whenfail="FAIL_DO_NOT_RETRY"/>
        <FlxbbDesign name="flxbb1" ncycles="3" constraints_sheet="10.0"
        sfxn_design="SFXN3" sfxn_relax="SFXN3" clear_all_residues="1"
        task_operations="limitchi2,layer_all" blueprint="./input2.bbskel"
        resfile="./input.resfile" />
        <Dssp name="dssp" />
        <FlxbbDesign name="flxbb2" ncycles="5" constraints_sheet="10.0"
        sfxn_design="SFXN3" sfxn_relax="SFXN3" clear_all_residues="1"
        task_operations="limitchi2,layer_all" blueprint="./input2.bbskel"
        resfile="./input.resfile" />

    </MOVERS>
    <APPLY_TO_POSE>
    </APPLY_TO_POSE>
    <PROTOCOLS>

        <Add mover_name="sse1"   />
        <Add mover_name="lover1" />
        <Add mover_name="sse2"   />
        <Add mover_name="lover2" />
       <Add mover_name="flxbb1" />
        <Add mover_name="dssp" />
        <Add filter_name="ss" />
        <Add filter_name="hp2" />
        <Add filter_name="sc" />
        <Add mover_name="flxbb2" />
        <Add mover_name="dssp" />
        <Add filter_name="ss" />
        <Add filter_name="pstat" />
        <Add filter_name="sc" />

    </PROTOCOLS>
</ROSETTASCRIPTS>
```

11 Sequence-based stability predictions of gcWGAN designs for 31 test folds and structure-based physical origins for 7 test cases

11.1 Stability

In vivo protein stability has been predicted based on protein primary sequences to compare the stability of generated, natural and random sequences. Specifically, we used instability index (Guruprasad et al., 1990), a measure widely used in literature and readily implemented in the software biopython (v.1.76) (Cock et al., 2009). A threshold of 40 is recommended for stability index, above which a protein is very likely unstable. For each of the 31 yielding test folds (with yield ratio at least 1E-5), up to 1,000 sequences were generated by the gcWGAN generator and the oracle filter under the running-time limit of 2 days. Random sequences were generated using uniform distribution of 20 standard amino acids at each position. The lengths and total

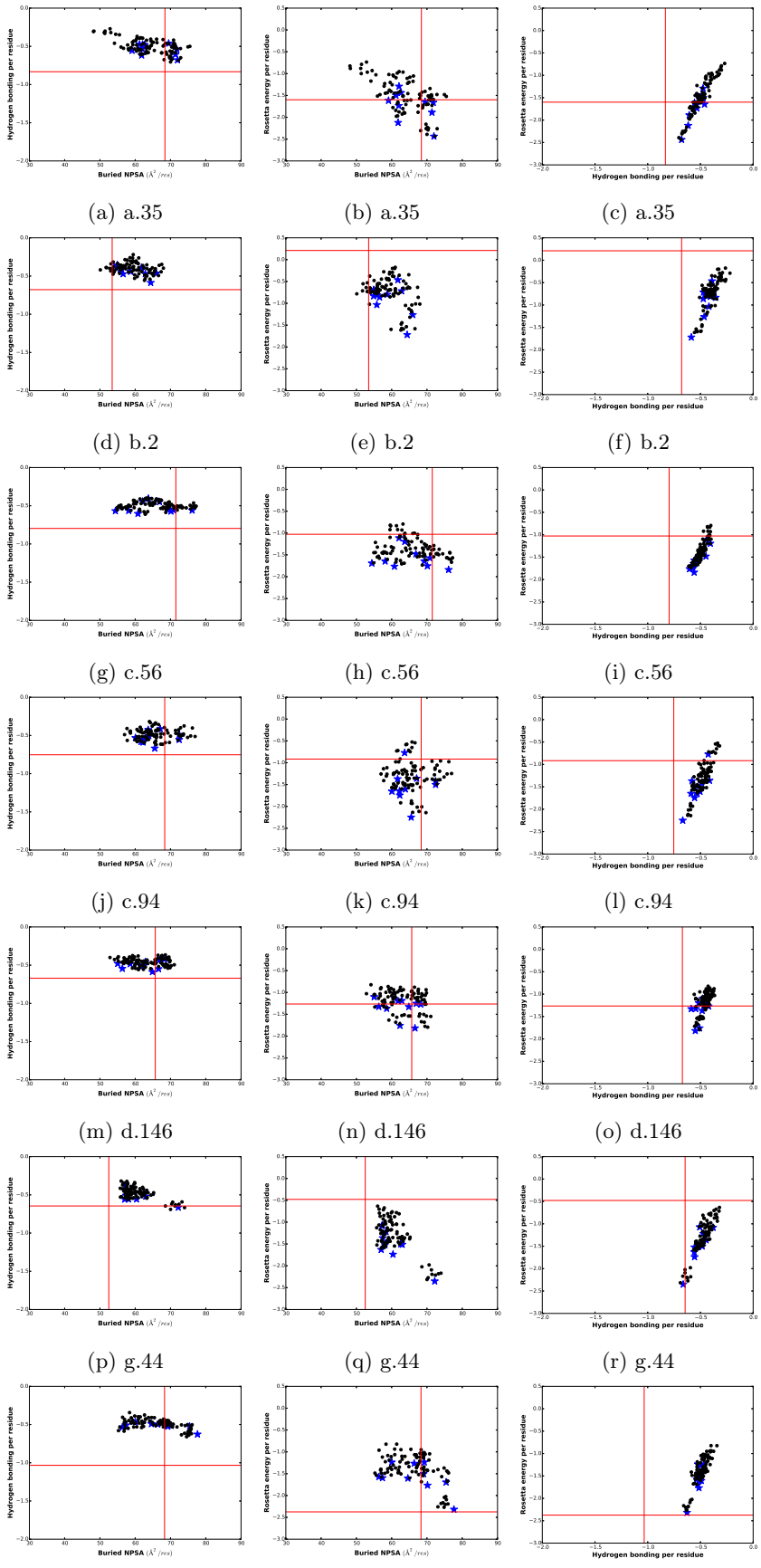
number of random sequences generated for each test fold were equal to the values in gcWGAN generated sequences.
Besides reporting the distributions of instability indices of designed, natural, and random sequences in the main text, we also calculated their means and pairwise distances (measured by Jensen-Shannon distances or JSD) below. One-sided Kolmogorov-Smirnov test suggested that the instability indices of gcWGAN designs are lower in distribution compared to random sequences, with a P-value of 2.14E-18.

Average	Random	gcWGAN	Natural
	45.49	39.63	39.88
Distance	JSD(rand,natural)	JSD(gcWGAN,natural)	JSD(rand,gcWGAN)
	0.75	0.45	0.83

Table S16: Comparison of gcWGAN generated sequences with random ones for stability

11.2 Stability origins

Rocklin et al. (2017) showed that Buried Non-polar surface area (NPSA), hydrogen bonding energy and Rosetta energy correlate well with stability. Therefore, we calculated these biophysical properties for both Rosetta-predicted structures of gcWGAN designs and ground-truth representative structures, in all 7 test folds for case studies. The results is presented in Fig. S16. Based on the figures, these biophysical properties correlate well with each other (Buried NPSA higher the better; Hydrogen bonding lower the better, Rosetta energy lower the better). Red lines in the subfigures show the corresponding biophysical property for the ground-truth representative protein for the desired fold. Blue asterisks represent the lowest Rosetta energy structures chosen for every sequence.



(a) a.35 (b) a.35 (c) a.35

(d) b.2 (e) b.2 (f) b.2

(g) c.56 (h) c.56 (i) c.56

(j) c.94 (k) c.94 (l) c.94

(m) d.146 (n) d.146 (o) d.146

(p) g.44 (q) g.44 (r) g.44

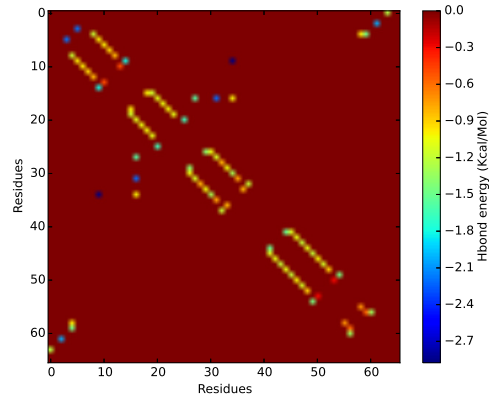
(s) novel (t) novel (u) novel

Figure S16: Calculation of biophysical properties related to stability for 7 case study folds

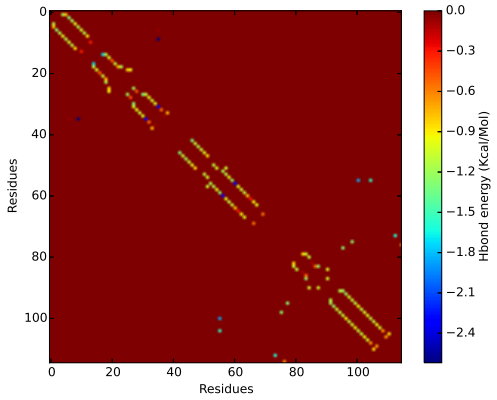
11.3 Fold origins

To understand why designed sequences could adopt a target fold, we have also compared the predicted hydrogen bond pattern for the protein representing the fold a.35 and a gcWGAN generated protein sequence (3D coordinates predicted through the Rosetta). Fold a.35 is an all alpha helix fold.
Based on the results shown in Figure S17, the predicted hydrogen bond for the gcWGAN generated sequence is similar to the expected alpha helix hydrogen bond pattern. To predict the hydrogen bond pattern, we perform residue energy breakdown based on Rosetta software with the following script and consider the sum of energy terms related to hydrogen such as 1) short range backbone hydrogen bonds 2) long range backbone hydrogen bonds 3) backbone-side chain hydrogen bonds 4) side chain-side chain hydrogen bonds.

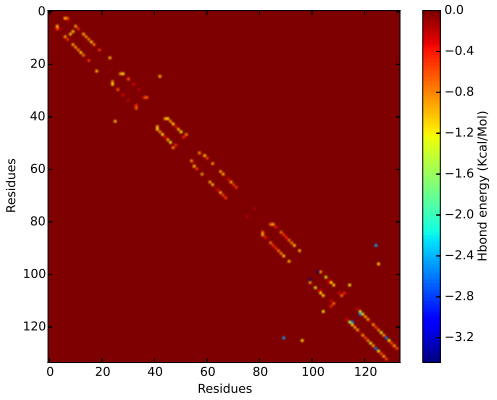
```
../../rosetta_bin_linux_2018.33.60351_bundle/main/source/bin/residue_energy_breakdown.linuxgccrelease \
-database ../../rosetta_bin_linux_2018.33.60351_bundle/main/database \
-in:file:s ${input_pdb} \
-out:file:silent ${output_file}
```



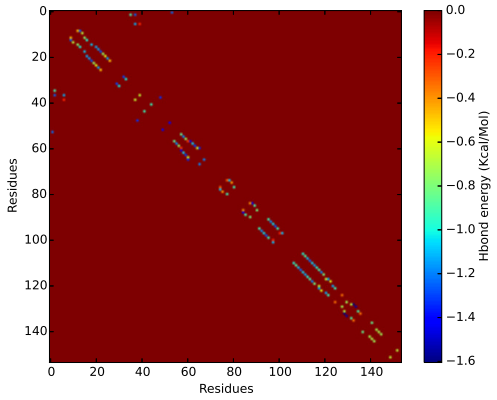
(a) Ground truth 1 (original one)



(b) Ground truth 2 (1s4k)

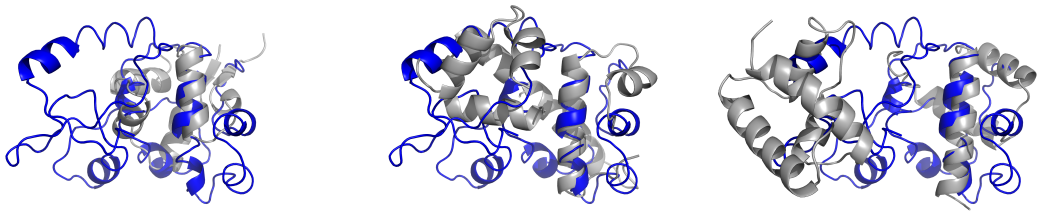


(c) Ground truth 3 (1cqt)



(d) gcWGAN generated

Figure S17: Comparison of predicted hydrogen bond pattern for protein representing the fold a.35 and the gcWGAN generated one for fold a.35.



(a) Ground truth 1 (original one) (b) Ground truth 2 (1s4k) (c) Ground truth 3 (1cqt)

Figure S18: Comparison of different structural alignments of gcWGAN generated sequence with 3 different members of fold a.35.

12 Sequence-based biophysical property predictions of gcWGAN designs for 31 test folds

We calculate and compare the biophysical properties of the proteins such as aromaticity value (Lobry and Gautier, 1994), grand average hydropathy (GRAVY) (Kyte and Doolittle, 1982), isoelectric point (pI) (Malamud and Drysdale, 1978), average flexibility index (Vihinen et al., 1994) and normalized molecular weight (Astbury and Woods, 1931) for generated sequence through gcWGAN, random and natural sequences. We utilized the biophysical properties implemented in software biopython package v.1.76 (Cock et al., 2009).
Like we did for stability prediction, for each of the 31 yielding test folds, we generated up to 1000 sequences with the running-time limit of 2 days. Random sequences were generated with a uniform distribution over 20 standard amino acids at each position and their lengths are the same as the gcWGAN-generated sequences. The total number of random sequences generated for each test fold is equal to that of gcWGAN generated ones as well.
Besides the distributions of generated, natural, and random sequences in these biophysical quantities, we also calculated the pairwise distances in their means or their distributions (measure by JSD again).

Biophysical property	$\lvert\mu_{\text{rand}} - \mu_{\text{nat.}}\rvert$	$\lvert\mu_{\text{gcWGAN}} - \mu_{\text{nat.}}\rvert$	JSD(rand,nat.)	JSD(gcWGAN,nat.)
Aromaticity value	0.076	0.007	0.832	0.531
GRAVY	0.196	0.157	0.701	0.486
Isoelectric point	0.406	0.136	0.788	0.666
Norm. molecular weight	7.359	1.574	0.790	0.602
Average flexibility index	0.010	0.002	0.818	0.493

Table S17: Comparison of gcWGAN generated sequences with random and natural sequences for biophysical properties

13 Biological significance of gcWGAN designs for 25 test folds

In order to demonstrate the biological significance of our generated sequences, we try to prove the consistency of the functional similarity, represented by Gene Ontology terms similarity (Ashburner et al., 2000), between the generated sequences and the representative sequences of the same fold.

Specifically, we represent the biological functions of each fold by the functions of the representative PDB of that fold. We use Gene Ontology terms to represent the biological functions and we retrieve the GO terms of those PDBs from GO-PDB association file from (Ashburner et al., 2000). As a result, we get 986 folds out of 1232 whose representative PDBs have experimentally validated functional annotations. In our testing set, 31 folds have non-zero yield ratios, within which 25 folds are within those 986 folds. Therefore, we will conduct our experiment on these 25 folds.

For each of these 25 folds, we generated 40 sequences through **gcWGAN** and our oracle filter ($25 \times 40 = 1000$ sequence in total). The GO terms of these 1,000 sequences were predicted by **NetGO** (You et al., 2019). Not all sequences were predicted to have GO terms on both ontologies of Molecular Function Ontology (**MFO**) and Biological Process Ontology (**BPO**). So we ended up having 840 gcWGAN sequences for MPO and 960 for BPO. For MPO or BPO, we calculated the GO similarity between each of these generated sequences and the representative PDB of the corresponding fold. The GO similarity is measured by GOGO (Zhao and Wang, 2018). Meanwhile, in order to set up a background threshold for each fold within 25 folds, we calculated the average GO similarity between that fold and the rest 986-1=985 folds. In the end, we calculated Delta GO similarity for each generated sequence, which is the similarity between the generated sequence and the representative PDB of the corresponding fold **minus** the background threshold of that fold. A Delta GO similarity above 0 indicates the fold specificity of the designs.

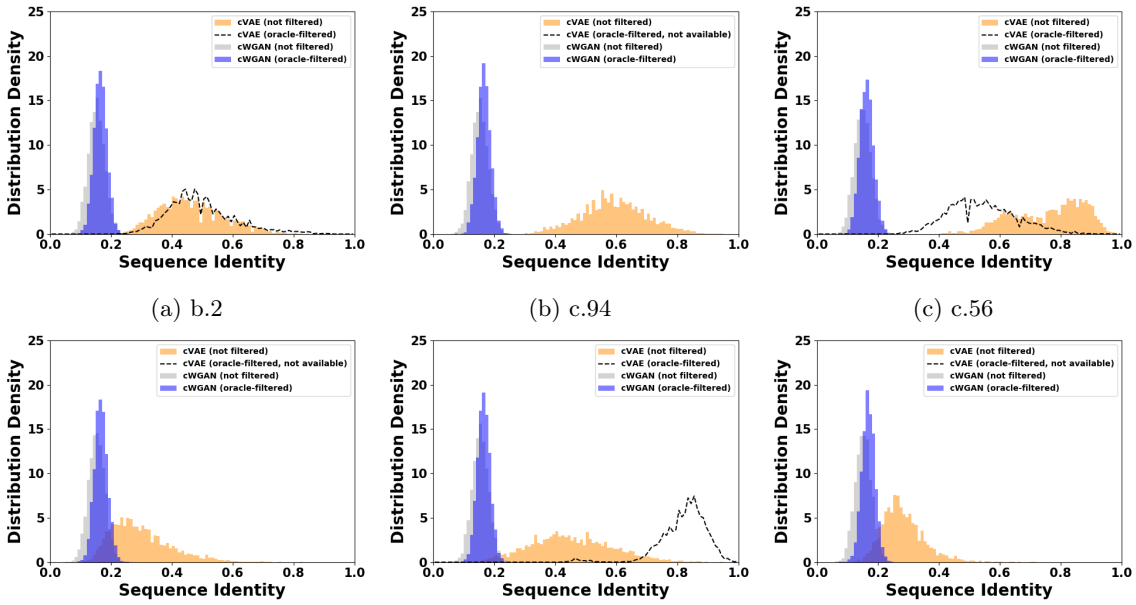
14 Assessing final sequence designs with case studies

For each of the six representative folds and a novel fold, we generated 100 valid sequences that pass the final check of the oracle (modified DeepSF). For each fold, we "folded" the first ten sequences, predicted 100 structures for each fold using Rosetta, and evaluated their structural accuracy compared to the known representative structure using TM-score (see main text).

Not only did we want to design sequences for the target folds, we also wanted to discover the sequence space that would be related to the folds. So for each of the 7 folds, we also calculated the pairwise identities among the generated sequences (sequence diversity) as well as that between the generated sequences and the natural ones (sequence novelty). Besides the previous generated sequences, we also applied our oracle on the cVAE-generated sequences and selected those passed the oracle for sequence diversity and novelty analysis. Since the yield ratio of some representative folds on cVAE is so low that it is infeasible to get enough sequences that pass the oracle. For some folds we generated more than 8 million sequences using cVAE but all of them failed to pass the oracle. So we could only do the filtered cVAE experiment for folds b.2, c.56 and g.44.

The diversity and novelty assessments were compared between gcWGAN and cVAE in the main text. Here we also include the comparison between cWGAN and cVAE. The following results show that, as gcWGAN, cWGAN designs (without oracle guidance) are also always more diverse and sometime more novel than cVAE designs. (The oracle feedback in gcWGAN would improve the yield ratio compared to cWGAN, as shown before.)

14.1 Diversity of sequences generated by cWGAN and cVAE



(a) b.2 (b) c.94 (c) c.56

(d) a.35 (e) g.44 (f) d.146

Figure S19: Sequence diversity for the 6 selected cases based on cWGAN

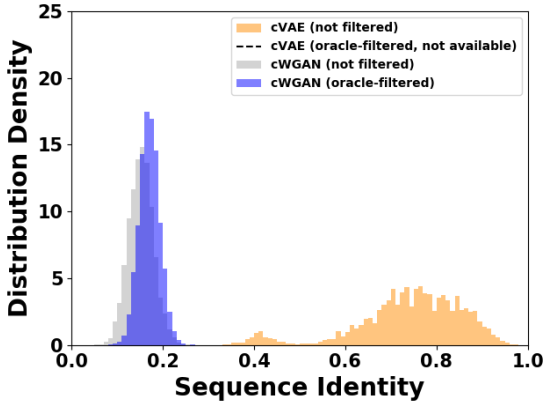
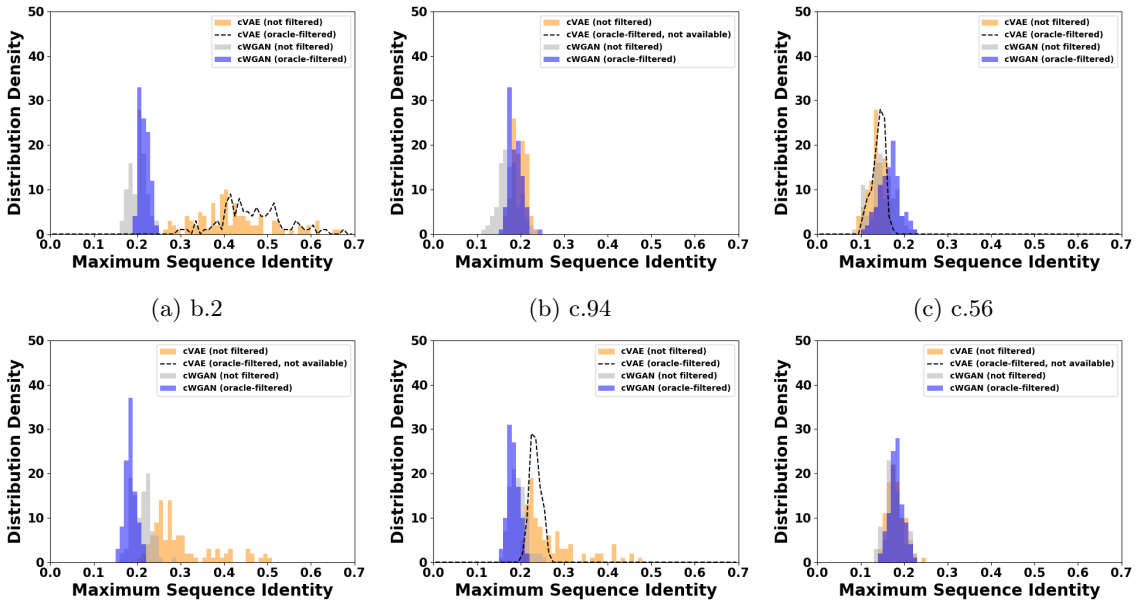


Figure S20: Sequence diversity for the novel fold based on cWGAN

We found that filters did not change the fact that cVAE designs were not diverse enough and contained many homolog pairs. It is rather the general Wasserstein distance that, by design, promoted the diversity of cWGAN or gcWGAN designs.

14.2 Novelty of sequences generated by cWGAN and cVAE



(a) b.2 (b) c.94 (c) c.56

(d) a.35 (e) g.44 (f) d.146

Figure S21: Sequence novelty distribution for the 6 selected cases based on cWGAN

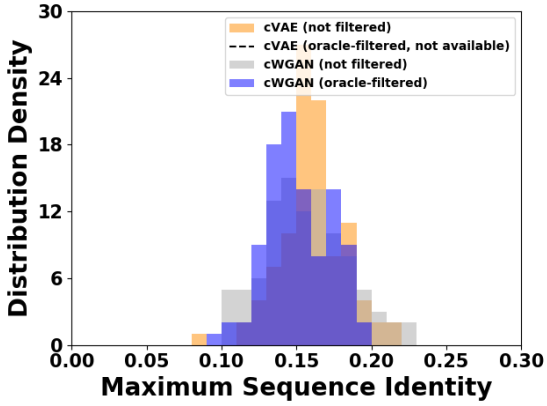


Figure S22: Sequence novelty distribution for the novel fold based on cWGAN

We found that, as gcWGAN, cWGAN designs are often more novel compared to cVAE regardless of applying the final oracle filter or not.

References

Ester, M.; Kriegel, H.; Sander, J.; Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD. 1996; pp 226–231.

Zhang, Y.; Skolnick, J. Scoring function for automated assessment of protein structure template quality. *Proteins: Struct., Funct., Bioinf.* **2004**, *57*, 702–710.

Zhang, Y.; Skolnick, J. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic Acids Res.* **2005**, *33*, 2302–2309.

Higham, N. J. Computing the nearest correlation matrix – a problem from finance. *J. Inst. Math. Its Appl.* **2002**, *22*, 329–343.

Hou, J.; Adhikari, B.; Cheng, J. DeepSF: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics* **2017**, *34*, 1295–1303.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. Advances in neural information processing systems. 2014; pp 2672–2680.

Karras, T.; Aila, T.; Laine, S.; Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196* **2017**,

Chu, C.; Zhmoginov, A.; Sandler, M. CycleGAN: a Master of Steganography. *arXiv preprint arXiv:1712.02950* **2017**,

Zhang, H.; Xu, T.; Li, H.; Zhang, S.; Huang, X.; Wang, X.; Metaxas, D. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint* **2017**,

Jin, Y.; Zhang, J.; Li, M.; Tian, Y.; Zhu, H.; Fang, Z. Towards the Automatic Anime Characters Creation with Generative Adversarial Networks. *arXiv preprint arXiv:1708.05509* **2017**,

Pathak, D.; Krahenbuhl, P.; Donahue, J.; Darrell, T.; Efros, A. A. Context encoders: Feature learning by inpainting. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016; pp 2536–2544.

Antipov, G.; Baccouche, M.; Dugelay, J. Face aging with conditional generative adversarial networks. Image Processing (ICIP), 2017 IEEE International Conference on. 2017; pp 2089–2093.

Yang, L.-C.; Chou, S.-Y.; Yang, Y.-H. MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847* **2017**,

Mirza, M.; Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* **2014**,

Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; Chen, X. Improved techniques for training gans. Advances in Neural Information Processing Systems. 2016; pp 2234–2242.

Arjovsky, M.; Chintala, S.; Bottou, b. p. y., L. Wasserstein generative adversarial networks.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. C. Improved training of wasserstein gans. Advances in Neural Information Processing Systems. 2017; pp 5767–5777.

Greener, J. G.; Moffat, L.; Jones, D. T. Design of metalloproteins and novel protein folds using variational autoencoders. *Sci. Rep.* **2018**, *8*, 16189.

Das, R.; André, I.; Shen, Y.; Wu, Y.; Lemak, A.; Bansal, S.; Arrowsmith, C. H.; Szyperski, T.; Baker, D. Simultaneous prediction of protein folding and docking at high resolution. *Proc. Natl. Acad. Sci.* **2009**, *106*, 18978–18983.

McGuffin, L. J.; Bryson, K.; Jones, D. T. The PSIPRED protein structure prediction server. *Bioinformatics* **2000**, *16*, 404–405.

Fleishman, S. J.; Leaver-Fay, A.; Corn, S. E. M., J. E.; Khare, S. D.; Koga, N.; Ashworth, J.; Murphy, P.; Richter, F.; Lemmon, G.; Meiler, J. RosettaScripts: a scripting language interface to the Rosetta macromolecular modeling suite. *PloS one* **2011**, *6*.

Koga, N.; Tatsumi-Koga, R.; Liu, G.; Xiao, R.; Acton, T. B.; Montelione, G. T.; Baker, D. Principles for designing ideal protein structures. *Nature* **2012**, *491*, 222–227.

Guruprasad, K.; Reddy, B. B.; Pandit, M. W. Correlation between stability of a protein and its dipeptide composition: a novel approach for predicting in vivo stability of a protein from its primary sequence. *Protein Eng., Des. Sel.* **1990**, *4*, 155–161.

Cock, P. J.; Antao, T.; Chang, J. T.; Chapman, B. A.; Cox, C. J.; Dalke, A.; Friedberg, I.; Hamelryck, T.; Kauff, F.; Wilczynski, B.; De Hoon, M. J. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **2009**, *25*, 1422–1423.

Rocklin, G. J.; Chidyausiku,; M., T.; Goreshnik, I.; Ford, A.; Houliston, S.; Lemak, A.; Carter, L.; Ravichandran, R.; Mulligan, V. K.; Chevalier, A.; Arrowsmith, C. H. Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science* **2017**, *357*, 168–175.

Lobry, J.; Gautier, C. Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 Escherichia coli chromosome-encoded genes. *Nucleic Acids Res.* **1994**, *22*, 3174–3180.

Kyte, J.; Doolittle, R. F. A simple method for displaying the hydropathic character of a protein. *J. Mol. Biol.* **1982**, *157*, 105–132.

Malamud, D.; Drysdale, J. W. Isoelectric points of proteins: a table. *Anal. Biochem.* **1978**, *86*, 620–647.

Vihinen, M.; Torkkila, E.; Riikonen, P. Accuracy of protein flexibility predictions. *Proteins: Struct., Funct., Bioinf.* **1994**, *19*, 141–149.

Astbury, W.; Woods, H. The molecular weights of proteins. *Nature* **1931**, *127*, 663–665.

Ashburner, M.; Ball, C. A.; Blake, J. A.; Botstein, D.; Butler, H.; Cherry, J. M.; Davis, A. P.; Dolinski, K.; Dwight, S. S.; Eppig, J. T.; Harris, M. A. Gene ontology: tool for the unification of biology. *Nat. Genet.* **2000**, *25*, 25.

You, R.; Yao, S.; Xiong, Y.; Huang, X.; Sun, F.; Mamitsuka, H.; Zhu, S. NetGO: improving large-scale protein function prediction with massive network information. *Nucleic Acids Res.* **2019**, *47*, W379–W387.

Zhao, C.; Wang, Z. GOGO: An improved algorithm to measure the semantic similarity between gene ontology terms. *Sci. Rep.* **2018**, *8*, 15107.