XENet: Using a new graph convolution to accelerate the timeline for protein design on quantum computers

Jack B. Maguire[1*], Daniele Grattarola[2], Eugene Klyshko[1,3], Vikram Khipple Mulligan[4], Hans Melo[1]

1 Menten AI, Inc., Palo Alto, CA, USA
2 Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland
3 Department of Physics, University of Toronto, Toronto, ON, Canada
4 Center for Computational Biology, Flatiron Institute, New York, NY, USA

* jbmaguire@menten.ai

Abstract

Graph representations are traditionally used to represent protein structures in sequence design protocols where the folding pattern is known. This infrequently extends to machine learning projects: existing graph convolution algorithms have shortcomings when representing protein environments. One reason for this is the lack of emphasis on edge attributes during massage-passing operations. Another reason is the traditionally shallow nature of graph neural network architectures. Here we introduce an improved message-passing operation that is better equipped to model local kinematics problems such as protein design. Our approach, XENet, pays special attention to both incoming and outgoing edge attributes.

We compare XENet against existing graph convolutions in an attempt to decrease rotamer sample counts in Rosetta's rotamer substitution protocol. This use case is motivating because it allows larger protein design problems to fit onto near-term quantum computers. XENet outperformed competing models while also displaying a greater tolerance for deeper architectures. We found that XENet was able to decrease rotamer counts by 40% without loss in quality. This decreased the problem size of our use case by more than a factor of 3. Additionally, XENet displayed an ability to handle deeper architectures than competing convolutions.

Author summary

Graphs data structures are ubiquitous in the field of protein design and are at the core of the recent advances in artificial intelligence brought forth by graph neural networks (GNNs). GNNs have led to some impressive results in modeling protein interactions, but are not as common as other tensor representations.

Most GNN architectures tend to put little to no emphasis on the information stored on edges; however, protein modeling tools often use edges to represent vital geometric relationships about residue pair interactions. In this paper, we show that a more advanced processing of edge attributes can lead to considerable benefits when modeling chemical data.

We introduce XENet, a new member of the GNN family that is shown to have improved ability to model protein residue environments based on chemical and geometric data. We use XENet to intelligently simplify the optimization problem that is solved when designing proteins. This task is important to us and others because it allows larger proteins to be designed on near-term quantum computers. We show that XENet is able to train on our protein modeling data better than existing methods, successfully resulting in a dramatic decrease in protein design sample space with no loss in quality.

Introduction

Protein design involves astronomically large search problems beyond the capabilities of even the largest supercomputers. [1] Current computational methods make use of stochastic search algorithms such as simulated annealing to handle this large space. [2] This task traditionally involves assuming a static protein backbone and representing all candidate sidechain conformations and identities as discrete possibilities called “rotamers”. [3, 4] A single sequence position on the protein can have hundreds of candidate rotamers when spanning all twenty native amino acids.

Quantum computing offers a new alternative for solving these complex tasks to power the development of new protein-based therapeutics and enzymes of industrial interest. [5] In previous work, we demonstrated how the protein design problem can be expressed as a combinatorial optimization problem and solved using quantum annealing hardware and hybrid quantum-classical solvers. [6] Critically, we were able to show the system’s applicability to real-world protein design problems without reducing the complexity of the problem.

This method used the Rosetta software suite to model these backbone-dependent rotamers and to calculate the one- and two-body interactions between them [7–9]. Our goal was to find the set of rotamers that minimizes the protein’s computed energy, measured in Rosetta Energy Units (REU). Rosetta does this using simulated annealing, in a process called “packing” and “rotamer substitution” [4, 10].

Mapping large protein design problems directly to quantum hardware was limited by a number of factors including noise and the number of qubits available. Even using a hybrid solver proved impractical for large problems as noise and time constraints effectively placed an upper barrier to the size of problems that could be solved. Additionally, we have evidence that the modeling of some atomic interactions, like hydrogen bonds, would be improved with a finer granularity of rotamer sampling, suggesting that our problem has reason to grow even larger [11].

Our goal for this project was to use machine learning to adaptively decrease sample space for arbitrary protein design problems by eliminating rotamers from consideration. Scientists are having rapidly-increasing success using artifical neural networks to design proteins using a variety of representations [12, 13]. We have recently seen success representing proteins by passing contact maps into image-inspired 2D convolutions [14, 15], 3D convolutions on voxelized representations [16, 17], and even language models on protein sequences [18–20]. The representation that interests us the most is the graph-based representation found in graph neural networks [21–23].

Graphs are intuitive representations for protein modeling cases in which the backbone structure is already established, as it is in protein design. In fact, traditional protein modeling tools such as Rosetta use graphs internally to model interactions during their own protocols [7, 24–26]. These residue-centric graphs represent each sequence position as a node, with edges connecting positions that are close in 3D space. Node attributes generally encode the residue’s backbone geometry and possibly some representation of its sidechain identity. Edge attributes are used to model the interactions and geometry between residue positions.

Graph neural networks (GNNs) are a class of machine learning models designed to process graph-structured data. While the seminal research on GNNs dates back to the works of Sperduti *et al.* [27], Gori *et al.* [28], and Scarselli *et al.* [29], recent research efforts have led to a rapid growth of the field and have achieved state-of-the-art results on a large variety of applications, ranging from social networks [30–32], to chemistry [33, 34], biology [21, 35, 36], and physics [37].

The growth of the field has led to the development of many diverse GNN architectures, notably including the works in references [38–43]. Of particular interest to this work are those models that can be expressed as message-passing architectures [44]. In particular, message-passing GNNs act on the node attributes of a graph according to the following general scheme:

$$\mathbf{x}'_i = \gamma\left(\mathbf{x}_i, \square_{j \in \mathcal{N}(i)}\, \phi\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{(j,i)}\right)\right), \quad \forall i \in \mathcal{V} \tag{1}$$

where ϕ is a *message* function that depends on the graph's node and edge attributes (resp. $\mathbf{X}$ and $\mathbf{E}$), $\square$ is any permutation-invariant operation that aggregates messages coming from the neighborhood of i, and γ is an *update* function (see our Notation section on the next page for the remaining symbols). Intuitively, message-passing GNNs transform the attributes of the graph by exchanging information between neighboring nodes.

While the definition of Eq. (1) allows the message function to depend on the edge attribute between a node and its neighbor, the majority of GNN architectures are designed for non-attributed edges. Among those GNNs that are designed to process edge attributes, we mention the Edge-Conditioned Convolutions (ECCs) introduced by Simonovsky and Komodakis [45]. ECCs make use of an auxiliary model called a *filter-generating network* (FGN) that takes as input edge attributes and produces output parameters that replace what conventionally would be the learnable parameters of ϕ in Eq. (1) that would ordinarily be fixed. ECCs can bring significant advantages when processing graphs for which edge attributes are important and have been used to process molecular graphs [46]. However, the FGN can be difficult to train due to the absence of a strong supervision signal (which is particularly difficult to achieve when stacking many layers) and ECCs are mostly effective in processing edge attributes with a one-hot representation.

In recent years, other types of GNNs have been proposed that process edge attributes directly in the message function, without relying on a FGN. These usually concatenate [47] or sum [48] the edge attributes to the node attributes of the neighbors. In particular, here we consider the work of Xie *et al.* [47], based on concatenation, which we denote as *CrystalConv* in the following.

We note, however, that all of the methods mentioned above suffer from two key issues. First, none of them are designed to take into account the case of symmetric directed graphs with asymmetric edge attributes (*i.e.*, graphs for which the existence of edge (i, j) implies the existence of edge (j, i) and *vice versa*, but the corresponding attributes can differ). This is particularly relevant for our work due to the geometric nature of our edge attributes: our edges themselves have no directionality but nearly every edge feature has some degree of asymmetry. Second, most existing methods are not designed to update edge attributes, which are considered as static inputs throughout the network. The updating of edge attributes is not a novel idea *per se*, since it was proposed both in the Graph Network model by Battaglia *et al.* [49] and in the Typed Graph Network of Prates *et al.* [50] (where both are works that attempt to unify GNNs in a similar spirit to the message passing framework), but to the best of our knowledge it is seldom applied in practice.

Here we propose XENet, a GNN model that addresses both concerns while also avoid the computational issues introduced by FGNs. XENet is a message-passing GNN that simultaneously accounts for both the incoming and outgoing neighbors of each node, such that a node's representation is based on the messages it receives as well as those it sends. We demonstrate XENet's advantage over ECC and CrystalConv by testing their abilities to eliminate rotamer candidates in real-world protein design problems.

Materials and methods

Notation Let a graph be a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with node set $\mathcal{V} = \{1, \ldots, N\}$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ s.t. $(i, j) \in \mathcal{E}$ is a directed edge from node i to node j. Additionally, let $\mathbf{x}_i \in \mathbb{R}^F$ indicate a vector attribute associated with node i and let $\mathbf{e}_{i,j} \in \mathbb{R}^S$ indicate a vector attribute associated with edge (i, j). We indicate the neighborhood of a node with $\mathcal{N}(i) = \{j \mid (j, i) \in \mathcal{E}\}$. Note that in our case we consider symmetric directed graphs, so that the incoming and outgoing neighbors of a node coincide.

To make notation more compact, in the following we denote with $\mathbf{X} \in \mathbb{R}^{N \times F}$ the matrix of node attributes, with $\mathbf{E} \in \mathbb{R}^{N \times N \times S}$ the matrix of edge attributes (we assume the entries of this matrix to be zero if the corresponding edge does not exist), and with $\mathbf{A} \in \{0, 1\}^{N \times N}$ the binary adjacency matrix of the graph.

XENet

Our architecture, which we refer to as XENet (due to its ability to convolve over both $\mathbf{X}$ and $\mathbf{E}$ tensors), is described by the following Equations:

$$\mathbf{s}_{ij} = \varphi^{(s)}\left(\mathbf{x}_i \| \mathbf{x}_j \| \mathbf{e}_{(i,j)} \| \mathbf{e}_{(j,i)}\right) \tag{2}$$

$$\mathbf{s}_i^{(\text{out})} = \sum_{j \in \mathcal{N}(i)} a^{(\text{out})}(\mathbf{s}_{ij}) \cdot \mathbf{s}_{ij} \tag{3}$$

$$\mathbf{s}_i^{(\text{in})} = \sum_{j \in \mathcal{N}(i)} a^{(\text{in})}(\mathbf{s}_{ij}) \cdot \mathbf{s}_{ji} \tag{4}$$

$$\mathbf{x}'_i = \varphi^{(n)}\left(\mathbf{x}_i \| \mathbf{s}_i^{(\text{out})} \| \mathbf{s}_i^{(\text{in})}\right) \tag{5}$$

$$\mathbf{e}'_{(i,j)} = \varphi^{(e)}\left(\mathbf{s}_{ij}\right) \tag{6}$$

where $\varphi^{(s)}$, $\varphi^{(n)}$, $\varphi^{(e)}$ are multi-layer perceptrons with Parametric Rectified Linear Unit activations [51], and where $a^{(\text{out})}$ and $a^{(\text{in})}$ are two dense layers with sigmoid activations and a single scalar output.

The core of XENet lies in the computation and aggregation of the *feature stacks* $\mathbf{s}_{ij}$ in Eqs. (2)-(4). These are obtained by concatenating the node and edge attributes associated with the incoming and outgoing messages (Eq. (2)), so that the multi-layer perceptron $\varphi^{(s)}$ learns to process the two directions separately. The feature stacks are also aggregated separately in the two directions of the flow, using self-attention [52] to compute a weighted sum (Eqs. (3)-(4)). The separate representations are concatenated and used to update the node attributes of the graph (Eq. (5)). Finally, some additional processing of the feature stacks through $\varphi^{(e)}$ lets us compute new edge attributes that are dependent on the message exchange between nodes (Eq. (6)).

Generating FixbbGCN Training Data

Here we prepare to apply XENet to a specific protein design problem, as described later in the paper. Our goal is to create a GNN that can analyze an intermediate protein state of FastDesign and predict which rotamers are likely to be sampled in the next round of rotamer substitution. We call this trained network “FixbbGCN”.

We used an arbitrary subset of structures from the Top8000 dataset for training [53], which ensures that no two protein structures have high similarity. Our training set used 967 structures (total of 229,776 residue positions) and our validation set used 239 structures (57,584 residue positions). The number of structures we used simply depended on how much CPU time we were willing to commit for generating data.

We ran 5 repeats of the MonomerDesign2019 variant of Rosetta's FastDesign [54] protocol on each structure but only collected training data for the final 4 repeats. We set Rosetta to generate a larger number of more finely-discretized rotamers by passing the '-ex1 -ex2' commandline flags and used Rosetta's REF2015 energy function [9]. This accounts for 16 of the 20 rounds of rotamer substitution, though for this project we only use the data from 4 of the 16 rounds due to score function ramping [54]. We therefore ended up with 919,104 training set elements (229,776 residues x 4 rounds per residue) and 230,336 validation elements.

For this project, rotamers from the 20 amino acids were binned into 54 categories. Alanine, Proline, and Glycine each had their own bin due to their lack of meaningful $\chi1$ attributes. The remaining 17 canonical amino acids had three bins each, which correspond to the three $\chi1$ wells.

For each round of rotamer substitution, we tracked the fraction of time that each rotamer was the representative state for its residue position. At the end of the run, any rotamer bin that held the representative state for more than 0.1% of the run was classified as a 1. All other rotamer bins were classified as a 0. Note that this resulted in a multi-label classification problem where every sample was associated with one or more classes. We also ignored data from the fraction of the simulated annealing trajectories where the simulated temperature was above 3 Rosetta Temperature Units (3 REU is intended to correspond with 3 kcal/mol).

FixbbGCN Architecture

We refer to this family of networks as FixbbGCN, as the Rosetta rotamer substitution protocol is sometimes called "fixbb". FixbbGCN is schematically represented in Fig 1. The model has three input tensors for **X**, **A**, and **E**. The maximum number of nodes per graph representation is $N = 30$, the number of attributes per node is $F = 46$, and the number of attributes per edge is $S = 28$. The output of the model is a 54-dimensional vector which holds one value for each of the rotamer bins described in the "Generating Training Data" section.

For all models, the **X** and **E** tensors are first fed to dense layers. These fully-connected layers only process one node/edge at a time, so that no information flows between nodes or edges. We then apply one or more steps of message passing, using either XENet, CrystalConv, or ECC layers. We used the Spektral package's implementation of the latter two layers. [55]

Fig 1 shows two rounds of message passing but we tested all models with one, two, and three layers (some XENet models were tested up to five layers, as reported in the SI). We note that the output tensor **E** from the final round of XENet is never be used by a future layer. The subset of parameters used to build this final **E** will be implicitly omitted when we tally trainable parameters.

We set FixbbGCN up as a single-node classification problem as opposed to a graph classification problem. Thus, after the message-passing stage, we focus on the unique node that represents the residue of interest being evaluated. We concatenate the output from the final message-passing layer with the original input **X** tensor in an effort to compensate the over-smoothing effect of message passing. We then crop the **X** tensor to only include the node that represents the protein residue of interest. FixbbGCN finishes off by running that single node's data through two more fully-connected layers.

All dense and message-passing layers have ReLU activation functions except for the final dense layer which has a sigmoid activation.

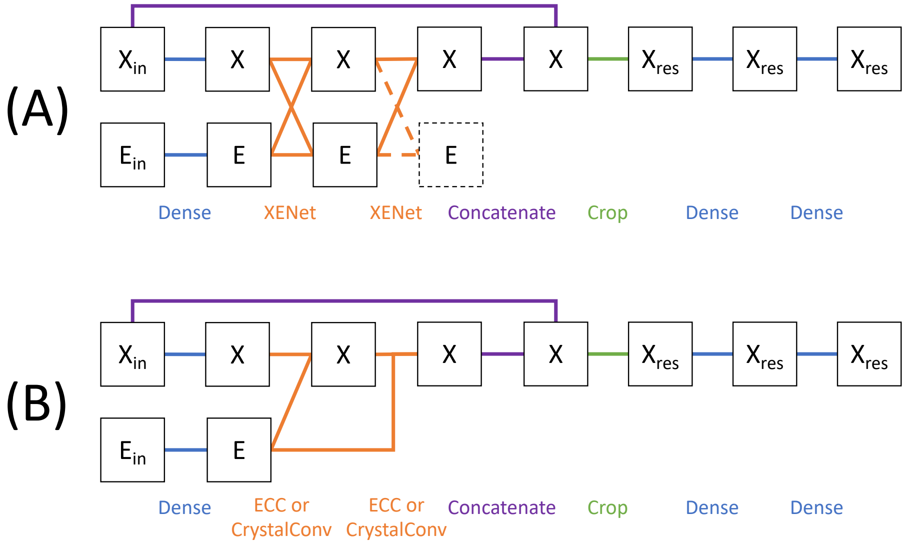

Fig 1. Schematic representation of FixbbGCNs, the networks used in our experiments. (A) Example layout for a model with two XENet layers. X denotes node attribute tensor with X_{in} as the input tensor and X_{res} as the single-node subset of the X tensor which represents the protein residue of interest. E denotes the edge attribute tensor with E_{in} as the input tensor. Dotted lines are used to represent operations that are omitted as described in the main text. (B) Example layout for a model with two ECC or CrystalConv layers using the same notation. The A tensor is omitted from this diagram because it never changes.

Hidden Layer Sizes

We benchmarked two XENet candidates as outlined in Table 1. XENet (s) is sized to have the same hidden layer size as the ECC models. XENet (p) is sized to have the same number of trainable parameters as the ECC models. We tuned these parameters by changing F_h and S_h, which are the number of channels for the hidden X and E layers, respectively, before the cropping layer. The penultimate dense layer always has 100 channels and the final layer always has 54 channels.

Likewise, we benchmarked two CrystalConv models using the same normalization techniques. The parameter normalization was not perfect but we got as close as possible without varying hyperparameters between depths of the same type.

Each XENet layer always used two internal stacking layers with S_h channels each. In other words, the $\varphi^{(s)}$ multi-layer perceptrons always had a depth of two.

Node and Edge Attributes

Our input data had 46 node attributes and 28 edge attributes, all of which are listed in the Supporting Information. Most of these attributes are direct physical characteristics of residues and physical relationships of residue pairs. We also included more advanced analytics in the form of Rosetta score terms.

Many of these attributes require access to the pyrosetta package to compute. [56] These include the Rosetta score terms, hydrogen bond identification, and the residue pair “jump” measurements. A Rosetta “jump” describes the six-dimensional rigid body relationship between the coordinate frames of two protein residues based on their backbone atoms.

Convolution	# Layers	Parameters	F_h	S_h
ECC	1	100,067	49	32
ECC	2	181,701	49	32
ECC	3	263,335	49	32
CrystalConv (s)	1	31,271	49	32
CrystalConv (s)	2	44,109	49	32
CrystalConv (s)	3	56,947	49	32
CrystalConv (p)	1	109,283	125	64
CrystalConv (p)	2	188,033	125	64
CrystalConv (p)	3	266,783	125	64
XENet (s)	1	30,421	49	32
XENet (s)	2	47,625	49	32
XENet (s)	3	64,829	49	32
XENet (p)	1	92,928	128	64
XENet (p)	2	179,522	128	64
XENet (p)	3	266,116	128	64

Table 1. Hidden layer sizes and number of trainable parameters for all models. F_h is the number of channels for hidden X layers and S_h is the number of channels for hidden E layers.

MentenGCN Package

We have created a public Python package in an effort to make protein processing with GNNs more portable and easier to share. MentenGCN [57] has a library of tensor decorators that were used for this project to generate the **X**, **A**, and **E** input tensors directly from Rosetta's protein representation. The configuration class for the GNN used in this paper is available within the MentenGCN package under the name "`Maguire_Grattarola_2021`". Please refer to the Supporting Information section for more detail on how to access this feature.

Training and Evaluating FixbbGCN Models

Each model configuration was trained between 6 and 12 times, loosely depending on the amount of resources required to train each model. We show later that the performance of a given architecture generally has narrow variance so we did not see the need to expand this sampling.

Each model was trained using Keras's implementation of the Adam optimizer with a starting learning rate of 0.001 and the binary crossentropy loss function [58, 59]. The learning rate was reduced by a factor of 10 whenever the validation loss plateaued for 2 consecutive epochs (`min_delta=0.001`). Training was halted whenever the validation loss plateaued for 5 consecutive epochs. We evaluated all models with binary crossentropy and Receiver Operating Characteristic (ROC) area-under-curve (AUC) on our validation set.

Benchmarking FixbbGCN Implementation On Classical Computer

As we will show in the Results section, the best model observed was XENet (p) with 3 layers. We benchmarked the applicability of this model by using it alongside Rosetta's packing protocol on six backbones of various sizes. For each backbone, we ran each residue position through our model and compared the 54 final values against a tuneable cutoff. Rotamers were eliminated if the final value for their respective bin fell below the cutoff. We performed this benchmark with a range of cutoffs between 0 and 1. We also included a cutoff of -1.0 as a control (so that no rotamers were eliminated, since the sigmoid activation has a minimum of 0). The larger the cutoff, the more aggressively rotamers were eliminated. We ran each cutoff on each structure 10 times and tracked the final Rosetta score in units of Rosetta Energy Units (REU) where more negative is better.

The Protein Data Bank codes for the six backbones used for this benchmark are 1SFX, 1ECO, 1D4O, 1W2C, 1O4S, and 1PJ5 in order of increasing size. All six of these structures are also from the top8000 dataset [53] so they are expected to have low homology with the training and validation data used to train the model. Staying consistent with the training data collection, Rosetta built rotamers with the "-ex1 -ex2" commandline flags and used Rosetta's REF2015 score function [9].

Benchmarking FixbbGCN Implementation On Quantum Computer

This quantum benchmark used all of the same Rosetta parameters and FixbbGCN cutoffs as the classical benchmark. We could not fit the previous test cases on the quantum machine so we used a subset of the smallest problem (protein data bank code: 1SFX). We used Rosetta's LayerSelector tool to design the 10 residues in the core of the protein. [60] All other residue positions were held immutable, decreasing our maximum rotamer count from 63183 to 5686.

Our quantum rotamer sampling protocol was identical to that described in Mulligan *et al.* [6] Like the classical benchmark, we ran 10 annealing trajectories for each FixbbGCN cutoff and reported the mean and standard deviation across those 10 samples. We also measured Random Access Memory (RAM) usage for each problem size. The RAM usage is expected to scale quadratically with rotamer count due to the need to calculate all residue pair energies between neighboring sequence positions.

Results and Discussion

FixbbGCN Model Comparisons

Our goal for this test was to find the graph convolution that would best represent our protein modeling data. XENet is our attempt to engineer a new GNN layer that makes further use of the edge tensors, including updating their features as the result of the convolution. As baseline model for this experiment we considered ECC, since it is one of the first and most widely used GNNs designed to process edge attributes, and we compare it against different configurations of CrystalConv and XENet to ensure a fair comparison. XENet (s) and CrystalConv (s) are normalized by the channel depth of each hidden layer. XENet (p) and CrystalConv (p) are normalized by the trainable parameter count.

The models were tasked with a multi-label classification problem to predict which protein sidechain rotamers would be sampled at a given sequence position during a round of Rosetta's rotamer subsitution protocol with simulated annealing. [10] We see in Table 2 that the XENet models outperform their ECC and CrystalConv counterparts, although some of the CrystalConv models are in close competition with the best XENet models. In addition to having better loss and AUC scores, XENet convolutions appear to perform better with deeper architectures. XENet slightly improves when the third graph convolution layer is introduced, whereas ECC and CrystalConv exhibit a consistent drop in performance at that depth.

Convolution	# Layers	Loss	σ Loss	AUC	σ AUC	# Models
ECC	1	0.188	0.004	0.9772	0.0009	8
ECC	2	0.213	0.071	0.9674	0.0224	6
ECC	3	5.442	0.462	0.6248	0.0284	6
CrystalConv (s)	1	0.173	0.001	0.9807	0.0003	8
CrystalConv (s)	2	0.155	0.002	0.9844	0.0003	8
CrystalConv (s)	3	4.520	0.258	0.6872	0.0179	8
CrystalConv (p)	1	0.158	0.002	0.9837	0.0005	8
CrystalConv (p)	2	0.145	0.002	0.9865	0.0004	8
CrystalConv (p)	3	5.522	0.345	0.6238	0.0368	8
XENet (s)	1	0.155	0.001	0.9844	0.0003	10
XENet (s)	2	0.147	0.002	0.9860	0.0005	8
XENet (s)	3	0.143	0.001	0.9868	0.0002	8
XENet (p)	1	0.143	0.001	0.9869	0.0002	10
XENet (p)	2	0.137	0.002	0.9878	0.0004	12
XENet (p)	3	0.134	0.002	0.9883	0.0004	8

Table 2. Training Results. Mean binary crossentropy loss and mean AUC for trained models. σ denotes standard deviation. Lower loss values are considered better whereas higher AUC values are better.

The reasons for these differences in performance can be readily motivated by considering the differences between the models themselves. First, ECC's FGN is an indirect way of processing edge attributes and requires a strong supervision signal in order to be trained effectively, which may not be easy to attain especially within deeper architectures. Second, ECC was often shown to be most effective when processing data with one-hot encoded attributes [45, 46], which is not the case here.

Since CrystalConv does not use a FGN to process the edges, it does not have the same problems as ECC and its performance is more in line with XENet's. However, the asymmetric processing of XENet, paired with its ability to update edge attributes to obtain a richer representation, make it more suitable for this particular type of data and results in a better overall performance in all configurations.

We show in Fig 2 that XENet can even handle depths of 4 and 5 GNN layers. The additional layers did not give us an advantage in validation loss; however, deeper architectures will theoretically be more advantageous for use cases that require more expansive message passing than our benchmark. For this reason, the mere ability to handle deeper architectures may prove to be a strength of XENet. XENet did encounter occasional failures with the deeper architectures but the majority of deeper models finished with competitive validation losses. We did not test CrystalConv or ECC with architectures of 4 or 5 layers due to their lack of success with 3 layers.

Quantum FixbbGCN Benchmark

Now that we have these trained models, we want to see how much they can decrease the sizes our quantum annealing use cases. We wrapped the best model for each architecture in Rosetta rotamer-elimination machinery and named it FixbbGCN ("fixbb" is a popular name for Rosetta's fixed-backbone packing protocol).

We cannot run full-sized quantum benchmarks for the same reason that this project was motivated: our protein design benchmarks are too large to be run on the quantum computers. The best we can currently do is use FixbbGCN to design a subset of the protein on the quantum annealer and save the larger problems for the classical benchmark presented later in the article.

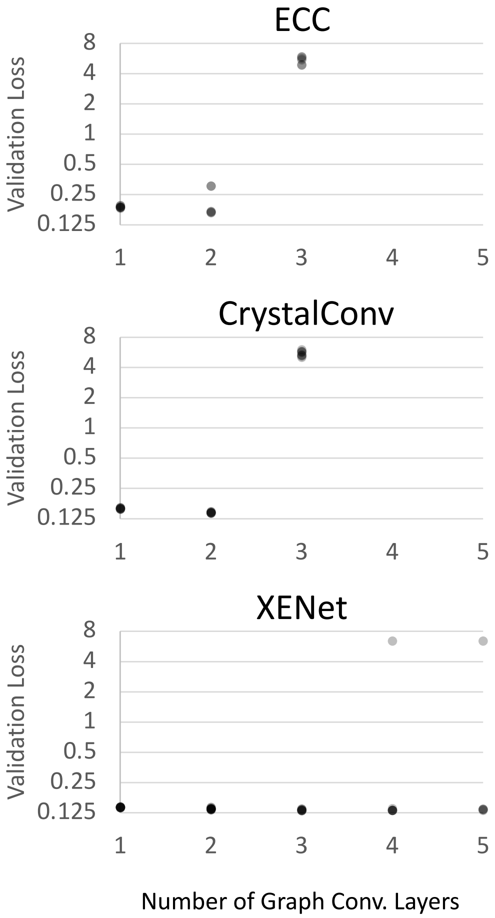

Fig 2. Depth Comparison With Fixed Parameter Count. We plot the losses of all trained ECC, CrystalConv (p), and XENet (p) models against the number of graph convolutional layers in each model. Transparency was applied to the points to help illustrate density. ECC and CrystalConv have no points with 4 or 5 layers.

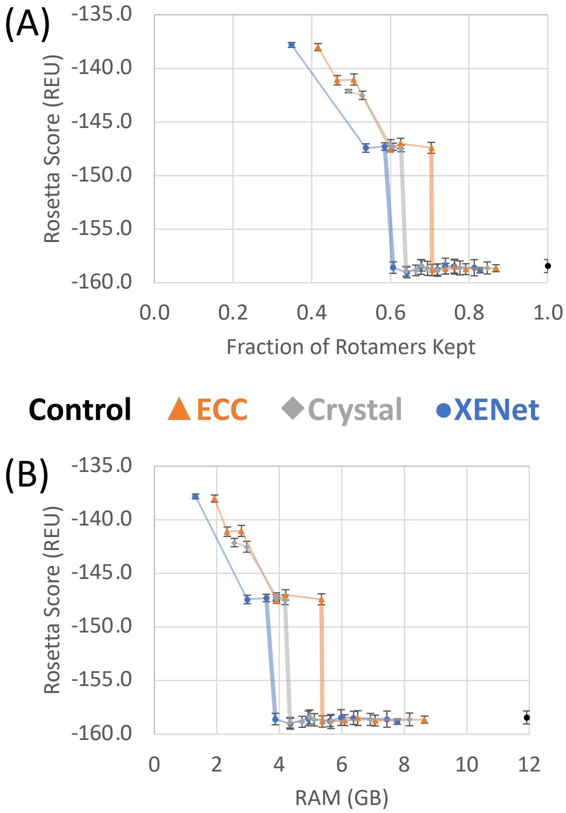

Fig 3. Quantum FixbbGCN Benchmark Results. (A) Mean Rosetta Scores for various cutoffs and convolutions types. Lines connect points of the same convolution and the line to the first drop in design quality is drawn thick. X-axis values are the number of surviving rotamers for a cutoff/convolution pair divided by the number of rotamers in the control case. (B) Same results as (A) but plotting against annealer memory usage instead of rotamer count. Both y-axes are truncated for the sake of readability.

For this test, we needed a very small problem size. We took the smallest test case from our benchmark set but restricted sampling to only include the core of the protein. We used Rosetta's definition of the core, which identified 10 residue positions that were sufficiently isolated from solvent exposure.

We chose this benchmark because the core is the most combinatorially challenging part of the protein to design. Rosetta samples core rotamers more finely than solvent-exposed residues so the rotamer count per position is higher. Additionally, these residue positions tend to have more neighbors, resulting in a more complex energy optimization problem.

XENet shows in Fig 3 an ability to decrease the rotamer count to roughly 60% before the dip in Rosetta score appears. ECC drops in quality near 70% and CrystalConv drops near 64%.

We did not report runtime for this benchmark because we had no way to decouple time spent running the annealer from time spent sending our data over the internet and waiting in the quantum computer's queue. We do expect that runtime will correlate linearly with RAM usage as both have quadratic relationships with the rotamer count.

Using RAM as our guide, XENet is able to reduce our problem size to 32% before the decrease in design quality appears. The CrystalConv model came close with a decrease to 36% and the ECC only model shrunk the problem to 43%.

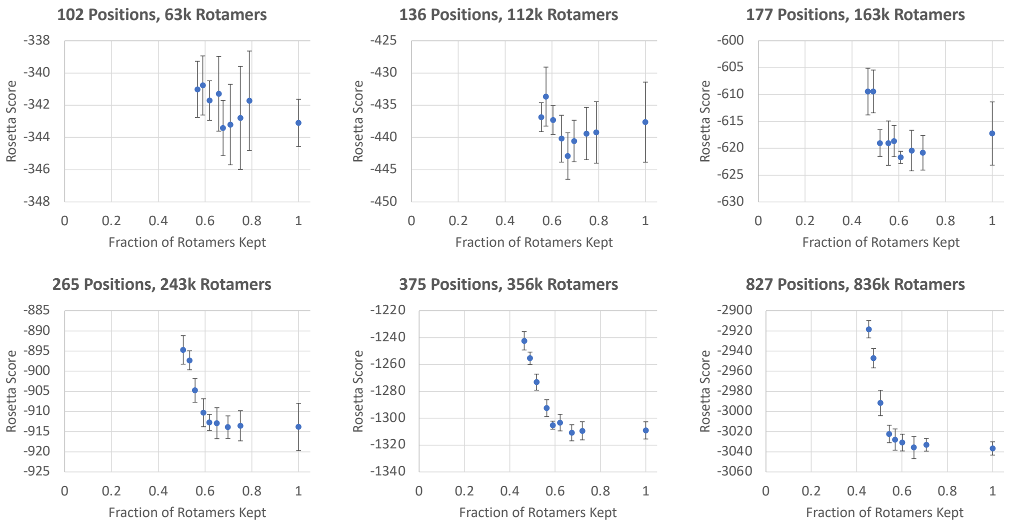

Fig 4. FixbbGCN Benchmark Results. Results of running Rosetta's rotamer substitution protocol on six different protein backbones. FixbbGCN was used with various cutoffs to decrease the total rotamer count of each sample. The mean Rosetta scores (measured in REU) and standard deviations are displayed for each cutoff. The y-axes are truncated for the sake of readability.

Classical FixbbGCN-XENet Benchmark

The goal for the final benchmark was to assess to what extent XENet's pattern observed in the quantum benchmark persists for full-sized use cases. Unfortunately, these full-sized design cases are too large for us to run on quantum computers so we ran these benchmarks using Rosetta's simulated annealer. This is the best we can do with current technology but hopefully a more complete test will be possible someday.

Similar to the quantum benchmark, this benchmark applies the XENet classifier with various cutoffs to Rosetta's set of rotamers for six different protein design problems. This time, however, the entire protein structures are being designed. Rotamers are pruned if their predicted value from the classifier is below the cutoff. The "control" data point with the largest rotamer count for a given use case is the standard Rosetta packing protocol with no influence from the classifier.

We see in Fig 4 that we can use FixbbGCN to decrease the number of rotamers without a loss in design quality to a limited extent. The Rosetta score will generally stay in range of the control data down to the range of 55-60% of the original rotamer count.

The results in Fig 4 supports the idea that FixbbGCN's ability to eliminate rotamers for small problem sizes translates to larger problem sizes too. It is up to the user to decide how risky they want to be with FixbbGCN, but our results suggests that decreasing rotamer counts to roughly 60% is safe.

Conclusion

Graph neural networks have great potential for modeling residue-level protein interactions. We show that our new convolution, XENet, can model residue-level environments better than existing methods ECC and CrystalConv. Not only does the usage of XENet result in lower validation losses, but we show that XENet can withstand deeper architectures.

To demonstrate XENet's value, we use it to create a tool capable of fitting larger protein design problems onto quantum computers by eliminating sidechain conformations that are unlikely to be selected by an annealer. XENet was consistently able to reduce rotamer counts by 40% without loss in design quality. As a result, we measured a 68% decrease in total problem size, which has a quadratic relationship with rotamer count.

Supporting information

Abbreviations

AUC	Area Under Curve (used with respect to ROC)
ECC	Edge-Conditioned Convolution
FGN	Filter-Generating Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
RAM	Random Access Memory
REU	Rosetta Energy Units
ROC	Receiver Operating Characteristic

Quantum Benchmark Results

XENet

Cutoff	Rotamers	Frac. Rotamers	RAM (GB)	Score (REU)	σ Score
-1	5686	1.00	11.92	-158.4	0.6
0.25	4706	0.83	7.78	-158.8	0.3
0.5	4623	0.81	7.45	-158.6	0.8
0.75	4334	0.76	6.38	-158.5	0.6
0.85	4206	0.74	5.98	-158.4	0.7
0.9	4089	0.72	5.62	-158.8	0.5
0.95	3851	0.68	4.92	-158.5	0.7
0.97	3649	0.64	4.35	-159.0	0.5
0.98	3452	0.61	3.87	-158.6	0.5
0.99	3331	0.59	3.58	-147.3	0.3
0.995	3057	0.54	2.97	-147.4	0.4
0.9995	1984	0.35	1.32	-137.8	0.2

Table S1. Results of Core Redesign on the quantum computer with XENet. σ denotes standard deviation.

ECC

Cutoff	Rotamers	Frac. Rotamers	RAM (GB)	Score (REU)	σ Score
-1	5686	1.00	11.92	-158.4	0.6
0.25	4937	0.87	8.65	-158.6	0.3
0.5	4500	0.79	7.07	-158.7	0.5
0.6	4340	0.76	6.51	-158.5	0.7
0.7	4208	0.74	6.09	-158.7	0.5
0.74	4013	0.71	5.38	-158.8	0.5
0.75	4002	0.70	5.35	-147.4	0.5
0.85	3559	0.63	4.20	-147.0	0.5
0.9	3408	0.60	3.91	-147.5	0.3
0.95	2884	0.51	2.79	-141.0	0.5
0.97	2644	0.47	2.33	-141.1	0.4
0.98	2365	0.42	1.93	-138.0	0.3

Table S2. Results of Core Redesign on the quantum computer with ECC. σ denotes standard deviation.

CrystalConv

Cutoff	Rotamers	Frac. Rotamers	RAM (GB)	Score (REU)	σ Score
-1	5686	1.00	11.92	-158.4	0.6
0.25	4812	0.85	8.16	-158.6	0.6
0.75	4416	0.78	6.91	-158.6	0.6
0.85	4094	0.72	5.65	-158.8	0.7
0.9	3947	0.69	5.12	-158.7	0.6
0.91	3865	0.68	4.98	-158.4	0.7
0.92	3779	0.66	4.73	-158.8	0.5
0.93	3637	0.64	4.34	-158.9	0.5
0.95	3566	0.63	4.18	-147.5	0.5
0.97	3449	0.61	3.91	-147.3	0.3
0.98	3416	0.60	3.91	-147.1	0.3
0.99	3008	0.53	2.96	-142.5	0.5
0.995	2805	0.49	2.56	-142.1	0.4
0.9995	1515	0.27	0.81	-131.8	0.1

Table S3. Results of Core Redesign on the quantum computer with CrystalConv. σ denotes standard deviation.

Node and Edge attributes for FixbbGCN

```
46 Node Features:
1 : 1 if the node is the focus residue, 0 otherwise
2 : 1 if residue is A, 0 otherwise
3 : 1 if residue is C, 0 otherwise
4 : 1 if residue is D, 0 otherwise
5 : 1 if residue is E, 0 otherwise
6 : 1 if residue is F, 0 otherwise
7 : 1 if residue is G, 0 otherwise
8 : 1 if residue is H, 0 otherwise
9 : 1 if residue is I, 0 otherwise
10 : 1 if residue is K, 0 otherwise
11 : 1 if residue is L, 0 otherwise
12 : 1 if residue is M, 0 otherwise
13 : 1 if residue is N, 0 otherwise
14 : 1 if residue is P, 0 otherwise
15 : 1 if residue is Q, 0 otherwise
16 : 1 if residue is R, 0 otherwise
17 : 1 if residue is S, 0 otherwise
18 : 1 if residue is T, 0 otherwise
19 : 1 if residue is V, 0 otherwise
20 : 1 if residue is W, 0 otherwise
21 : 1 if residue is Y, 0 otherwise
22 : Phi of the given residue, measured in radians.
 Spans from -pi to pi
23 : Psi of the given residue, measured in radians.
 Spans from -pi to pi
24 : Sine of chi angle number 1 of each residue.
 Spans from -1 to 1, 0 if chi angle is not valid for this residue
```

```
25 : Cosine of chi angle number 1 of each residue.
 Spans from -1 to 1, 0 if chi angle is not valid for this residue
26 : Sine of chi angle number 2 of each residue.
 Spans from -1 to 1, 0 if chi angle is not valid for this residue
27 : Cosine of chi angle number 2 of each residue.
 Spans from -1 to 1, 0 if chi angle is not valid for this residue
28 : Sine of chi angle number 3 of each residue.
 Spans from -1 to 1, 0 if chi angle is not valid for this residue
29 : Cosine of chi angle number 3 of each residue.
 Spans from -1 to 1, 0 if chi angle is not valid for this residue
30 : Sine of chi angle number 4 of each residue.
 Spans from -1 to 1, 0 if chi angle is not valid for this residue
31 : Cosine of chi angle number 4 of each residue.
 Spans from -1 to 1, 0 if chi angle is not valid for this residue
32 : fa_atr onebody term using ref2015
33 : fa_rep onebody term using ref2015
34 : fa_sol onebody term using ref2015
35 : fa_intra_rep onebody term using ref2015
36 : fa_intra_sol_xover4 onebody term using ref2015
37 : lk_ball_wtd onebody term using ref2015
38 : fa_elec onebody term using ref2015
39 : pro_close onebody term using ref2015
40 : hbond_bb_sc onebody term using ref2015
41 : omega onebody term using ref2015
42 : fa_dun onebody term using ref2015
43 : p_aa_pp onebody term using ref2015
44 : yhh_planarity onebody term using ref2015
45 : ref onebody term using ref2015
46 : rama_prepro onebody term using ref2015

28 Edge Features:
1 : 1.0 if the two residues are polymer-bonded,
0.0 otherwise (symmetric)
2 : Euclidean distance between the CA atoms of each residue,
measured in Angstroms (symmetric)
3 : Euclidean distance between the two CB atoms of each residue,
measured in Angstroms (symmetric)
4 : CA-CB-CB-CA torsion angle in radians,
spans from -pi to pi (symmetric)
5 : N1-CA1-CB1-CB2 torsion angle in radians,
spans from -pi to pi (asymmetric)
6 : CA1-CB1-CB2 bond angle in radians, spans from 0 to pi (asymmetric)
7 : Natural Log of the sequence distance between the two residues
(i.e., number of residues between these two residues
in sequence space, plus one). -1.0 if the two residues
belong to different chains. (symmetric)
8-10 : Translation vector for the Rosetta jump.
   Distances are measured in Angstroms (asymmetric)
11-13 : Euler angles for the Rosetta jump (asymmetric)
14 : Total number of backbone-backbone hydrogen bonds (symmetric)
15 : Total number of backbone-sidechain hydrogen bonds (symmetric)
16 : Total number of sidechain-sidechain hydrogen bonds (symmetric)
```

```
17 : Number of hydrogen bonds in which
 the first residue is the donor (asymmetric)
18 : Number of hydrogen bonds in which
 the first residue is the acceptor (asymmetric)
19 : fa_atr twobody term using ref2015 (symmetric)
20 : fa_rep twobody term using ref2015 (symmetric)
21 : fa_sol twobody term using ref2015 (symmetric)
22 : lk_ball_wtd twobody term using ref2015 (symmetric)
23 : fa_elec twobody term using ref2015 (symmetric)
24 : hbond_sr_bb twobody term using ref2015 (symmetric)
25 : hbond_lr_bb twobody term using ref2015 (symmetric)
26 : hbond_bb_sc twobody term using ref2015 (symmetric)
27 : hbond_sc twobody term using ref2015 (symmetric)
28 : dslf_fa13 twobody term using ref2015 (symmetric)
```

These attributes can be reproduced with the following python code

```
# pip install menten-gcn
import menten_gcn as mg

data_maker = mg.published.Maguire_Grattarola_2021()
data_maker.summary()

# This check ensures that the data_maker gives the expected values
data_maker.run_consistency_check()

# Visit https://menten-gcn.readthedocs.io/ to see
# how to use this data_maker with your protein
```

Raw Data

This section attempts to comply with PLOS Computational Biology's data policy. We provide all individual data points that are only summarized as means in the main text.

Downloadables

Raw training data is publicly available at https://menten-ai-public.s3.us-east-2.amazonaws.com/Maguire-XENet-2021/all_training_data.tar.gz

Raw testing data is publicly available at https://menten-ai-public.s3.us-east-2.amazonaws.com/Maguire-XENet-2021/all_testing_data.tar.gz

Our best ECC model (used for quantum benchmark) is available in Keras h5 format at https://menten-ai-public.s3.us-east-2.amazonaws.com/Maguire-XENet-2021/best_ECC.h5

Our best CrystalConv model (used for quantum benchmark) is available in Keras h5 format at https://menten-ai-public.s3.us-east-2.amazonaws.com/Maguire-XENet-2021/best_CrystalConv.h5

Our best XENet model (used for quantum and classical benchmarks) is available in Keras h5 format at https://menten-ai-public.s3.us-east-2.amazonaws.com/Maguire-XENet-2021/best_XENet.h5

Training Losses

Each table below lists the data points for the means and standard deviations reported in Table 2 (columns 3 and 4) of the main text.

ECC 1	ECC 2	ECC 3
0.1823	0.1642	4.8554
0.1855	0.1708	4.8884
0.1868	0.3053	5.5053
0.1868	0.1664	5.6768
0.1883	0.1675	5.8551
0.1889	0.3027	5.8732
0.1935		
0.1940		

CrystalConv (s) 1	CrystalConv (s) 2	CrystalConv (s) 3
0.171	0.153	4.203
0.172	0.153	4.327
0.172	0.154	4.379
0.173	0.156	4.398
0.173	0.156	4.481
0.173	0.156	4.599
0.174	0.157	4.809
0.175	0.157	4.967

CrystalConv (p) 1	CrystalConv (p) 2	CrystalConv (p) 3
0.156	0.142	5.047
0.157	0.143	5.209
0.157	0.143	5.316
0.158	0.144	5.319
0.158	0.145	5.690
0.160	0.146	5.770
0.161	0.146	5.789
0.162	0.148	6.035

XENet (s) 1	XENet (s) 2	XENet (s) 3
0.1534	0.1444	0.1413
0.1535	0.1445	0.1423
0.1540	0.1458	0.1425
0.1545	0.1459	0.1427
0.1549	0.1460	0.1428
0.1550	0.1461	0.1432
0.1551	0.1488	0.1434
0.1565	0.1516	0.1444
0.1566		
0.1577		

XENet (p) 1	XENet (p) 2	XENet (p) 3	XENet (p) 4	XENet (p) 5
0.1414	0.1346	0.1311	0.1325	0.1332
0.1415	0.1353	0.1327	0.1326	0.1349
0.1416	0.1353	0.1328	0.1326	0.1355
0.1420	0.1364	0.1333	0.1330	0.1361
0.1422	0.1365	0.1335	0.1330	6.4491
0.1426	0.1366	0.1340	0.1377	
0.1428	0.1366	0.1368	6.4531	
0.1431	0.1376	0.1372		
0.1436	0.1379			
0.1446	0.1388			
	0.1406			
	0.1419			

AUCs

Each table below lists the data points for the means and standard deviations reported in Table 2 (columns 5 and 6) of the main text.

ECC 1	ECC 2	ECC 3
0.9772	0.9826	0.6144
0.9779	0.9810	0.6173
0.9770	0.9379	0.6662
0.9760	0.9820	0.5860
0.9775	0.9818	0.6160
0.9757	0.9391	0.6488
0.9775		
0.9785		

CrystalConv (s) 1	CrystalConv (s) 2	CrystalConv (s) 3
0.9811	0.9844	0.6693
0.9805	0.9848	0.6788
0.9803	0.9841	0.6877
0.9809	0.9843	0.6851
0.9807	0.9846	0.6723
0.9805	0.9841	0.6891
0.9809	0.9848	0.6881
0.9805	0.9843	0.7274

CrystalConv (p) 1	CrystalConv (p) 2	CrystalConv (p) 3
0.9840	0.9865	0.6208
0.9833	0.9869	0.6587
0.9833	0.9861	0.6569
0.9838	0.9868	0.6434
0.9843	0.9858	0.5513
0.9841	0.9867	0.6494
0.9839	0.9866	0.6136
0.9829	0.9862	0.5962

XENet (s) 1	XENet (s) 2	XENet (s) 3
0.9846	0.9865	0.9869
0.9841	0.9861	0.9865
0.9846	0.9862	0.9867
0.9848	0.9862	0.9869
0.9847	0.9850	0.9867
0.9844	0.9861	0.9869
0.9842	0.9865	0.9870
0.9839	0.9857	0.9867
0.9844		
0.9845		

XENet (p) 1	XENet (p) 2	XENet (p) 3	XENet (p) 4	XENet (p) 5
0.9871	0.9881	0.9883	0.9885	0.9885
0.9867	0.9880	0.9885	0.9887	0.9882
0.9867	0.9872	0.9885	0.5893	0.5903
0.9866	0.9879	0.9889	0.9886	0.9881
0.9865	0.9882	0.9884	0.9885	0.9878
0.9868	0.9883	0.9885	0.9884	
0.9871	0.9884	0.9878	0.9877	
0.9870	0.9881	0.9876		
0.9870	0.9879			
0.9869	0.9874			
	0.9877			
	0.9869			

Quantum Benchmark Scores

ECC:

Cutoff	Scores									
0.25	-258.74	-258.32	-258.33	-258.71	-258.94	-258.79	-258.96	-258.11	-258.39	-259.09
0.5	-259.26	-259.26	-258.16	-258.71	-259.26	-258.81	-258.82	-258.87	-258.37	-257.75
0.6	-257.09	-258.51	-257.89	-258.59	-258.71	-258.89	-257.94	-259.35	-258.64	-259.35
0.7	-258.87	-258.82	-259.65	-258.69	-257.79	-258.59	-258.50	-258.14	-258.79	-259.04
0.74	-259.26	-258.42	-258.29	-259.09	-258.25	-259.52	-258.57	-258.13	-259.19	-259.39
0.75	-247.93	-247.02	-247.63	-246.94	-246.69	-246.71	-247.80	-247.93	-247.77	-247.73
0.85	-247.57	-247.31	-246.83	-246.10	-246.42	-247.15	-247.63	-246.80	-247.02	-247.21
0.9	-247.13	-247.36	-247.62	-247.94	-247.24	-247.45	-247.59	-247.26	-247.86	-247.76
0.95	-241.84	-241.09	-241.70	-240.38	-240.53	-241.51	-240.88	-240.59	-240.63	-241.24
0.97	-240.68	-241.59	-241.08	-241.19	-240.95	-240.51	-241.81	-240.96	-240.71	-241.61
0.98	-237.76	-238.11	-238.26	-238.34	-237.59	-237.48	-238.07	-237.88	-238.23	-238.45
0.99	-232.58	-232.35	-232.39	-232.40	-232.53	-232.09	-232.31	-232.47	-232.57	-231.97
0.995	-229.95	-230.16	-229.77	-230.00	-230.35	-229.65	-229.89	-229.67		
0.9995	-197.13	-196.87	-196.88	-196.86	-196.88	-196.87	-196.78	-197.05		

CrystalConv:

Cutoff	Scores									
0.25	-258.56	-259.01	-257.93	-258.32	-258.23	-259.09	-259.65	-259.05	-258.63	-257.70
0.75	-257.88	-258.64	-258.89	-259.57	-259.09	-258.26	-258.80	-258.25	-257.82	-258.97
0.85	-259.39	-257.46	-259.26	-259.26	-258.63	-258.66	-259.32	-259.26	-257.94	-258.96
0.9	-258.10	-258.06	-258.15	-258.18	-259.13	-258.64	-258.29	-259.52	-259.39	-259.09
0.91	-258.42	-258.84	-257.44	-259.26	-259.13	-258.67	-258.79	-257.63	-257.78	-257.89
0.92	-259.09	-258.29	-258.60	-258.64	-258.26	-259.65	-259.65	-259.17	-258.48	-258.64
0.93	-258.40	-259.05	-259.52	-258.59	-259.19	-258.66	-258.29	-259.39	-259.65	-258.52
0.95	-246.94	-246.56	-247.08	-247.72	-247.79	-247.86	-247.89	-247.45	-247.63	-247.75
0.97	-247.47	-247.49	-247.85	-247.00	-247.34	-247.22	-247.56	-247.07	-247.35	-246.79
0.98	-247.33	-247.35	-247.14	-246.76	-247.76	-247.31	-247.19	-246.59	-247.21	-246.87
0.99	-243.07	-242.06	-243.18	-242.85	-242.95	-242.72	-241.83	-242.52	-241.87	-242.12
0.995	-242.33	-241.28	-242.24	-241.93	-242.32	-242.53	-242.41	-242.44	-242.08	-241.72
0.9995	-231.85	-231.66	-231.63	-231.67	-231.98	-231.88	-231.80	-231.79	-231.90	-231.51

	Cutoff	Scores									
	-1	-258.30	-258.67	-258.52	-257.66	-258.68	-259.52	-258.23	-259.05	-258.31	-257.44
	0.25	-258.84	-258.89	-258.60	-259.26	-259.13	-258.89	-258.69	-258.89	-258.60	-258.42
	0.5	-258.98	-258.13	-259.06	-258.17	-259.26	-257.11	-259.01	-258.64	-259.65	-257.94
	0.75	-258.17	-258.60	-258.96	-258.66	-259.09	-258.41	-258.29	-257.17	-259.35	-258.11
	0.85	-258.87	-258.60	-258.30	-257.61	-258.69	-259.04	-259.57	-257.00	-258.60	-258.13
XENet:	0.9	-258.63	-258.08	-257.79	-259.07	-258.82	-258.50	-259.09	-259.26	-259.35	-259.09
	0.95	-258.30	-257.56	-257.92	-259.26	-259.32	-257.66	-258.11	-259.26	-258.79	-259.02
	0.97	-258.39	-258.89	-259.52	-258.46	-258.32	-258.95	-259.57	-259.65	-259.26	-259.26
	0.98	-258.73	-258.95	-258.03	-257.82	-257.93	-258.60	-258.64	-258.51	-259.26	-259.39
	0.99	-246.71	-246.96	-247.05	-247.67	-247.89	-247.47	-247.14	-247.48	-247.35	-247.23
	0.995	-246.80	-247.62	-247.86	-246.88	-247.89	-247.76	-247.17	-247.71	-247.25	-247.39
	0.9995	-237.68	-238.05	-238.02	-237.93	-238.09	-238.02	-237.62	-237.75	-237.56	-237.48

Note the -1 cutoff value from XENet table was used as the control for the entire experiment. No rotamers were eliminated with that cutoff so it is a global control.

Acknowledgments

We thank Dr. Andrew Leaver-Fay and Brian Coventry for various Rosetta developments that made our workflow easier and Sergey Lyskov and Dan Farrell for their assistance in overcoming technical hurdles. We also thank D-Wave Systems, Inc. for useful discussions and support.

References

1. Huang PS, Boyken SE, Baker D. The coming of age of de novo protein design. Nature. 2016;537(7620):320–327.

2. Kuhlman B, Bradley P. Advances in protein structure prediction and design. Nat Rev Mol Cell Biol. 2019;20(11):681–697.

3. Dunbrack RL. Rotamer libraries in the 21st century. Curr Opin Struct Biol. 2002;12(4):431–440.

4. Kuhlman B, Dantas G, Ireton GC, Varani G, Stoddard BL, Baker D. Design of a novel globular protein fold with atomic-level accuracy. Science. 2003;302(5649):1364–1368.

5. Outeiral C, Strahm M, Shi J, Morris GM, Benjamin SC, Deane CM. The prospects of quantum computing in computational molecular biology. 2020;doi:10.1002/wcms.1481.

6. Mulligan VK, Melo H, Merritt HI, Slocum S, Weitzner BD, Watkins AM, et al. Designing Peptides on a Quantum Computer. bioRxiv. 2020;doi:10.1101/752485.

7. Leaver-Fay A, Tyka M, Lewis SM, Lange OF, Thompson J, Jacak R, et al. ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules. Methods Enzymol. 2011;487:545–574.

8. Leman JK, Weitzner BD, Lewis SM, Adolf-Bryfogle J, Alam N, Alford RF, et al. Macromolecular modeling and design in Rosetta: recent methods and frameworks. Nat Methods. 2020;17(7):665–680.

9. Park H, Bradley P, Greisen P, Liu Y, Mulligan VK, Kim DE, et al. Simultaneous Optimization of Biomolecular Energy Functions on Features from Small Molecules and Macromolecules. J Chem Theory Comput. 2016;12(12):6201–6212.

10. Kuhlman B, Baker D. Native protein sequences are close to optimal for their structures. Proc Natl Acad Sci U S A. 2000;97(19):10383–10388.

11. Maguire JB, Boyken SE, Baker D, Kuhlman B. Rapid Sampling of Hydrogen Bond Networks for Computational Protein Design. J Chem Theory Comput. 2018;14(5):2751–2760.

12. Gao W, Mahajan SP, Sulam J, Gray JJ. Deep Learning in Protein Structural Modeling and Design. Patterns. 2020;1(9):100142. doi:10.1016/j.patter.2020.100142.

13. Yang KK, Wu Z, Arnold FH. Machine-learning-guided directed evolution for protein engineering. Nature Methods. 2019;16(8):687–694. doi:10.1038/s41592-019-0496-6.

14. Anishchenko I, Chidyausiku TM, Ovchinnikov S, Pellock SJ, Baker D. De novo protein design by deep network hallucination. bioRxiv. 2020;doi:10.1101/2020.07.22.211482.

15. Linder J, Seelig G. Fast differentiable DNA and protein sequence optimization for molecular design; 2020.

16. Anand-Achim N, Eguchi RR, Derry A, Altman RB, Huang PS. Protein sequence design with a learned potential. bioRxiv. 2020;doi:10.1101/2020.01.06.895466.

17. Zhang Y, Chen Y, Wang C, Lo CC, Liu X, Wu W, et al. ProDCoNN: Protein design using a convolutional neural network. Proteins: Structure, Function, and Bioinformatics. 2020;88(7):819–829. doi:https://doi.org/10.1002/prot.25868.

18. Xu Y, Verma D, Sheridan RP, Liaw A, Ma J, Marshall NM, et al. Deep Dive into Machine Learning Models for Protein Engineering. Journal of Chemical Information and Modeling. 2020;60(6):2773–2790. doi:10.1021/acs.jcim.0c00073.

19. Sabban S, Markovsky M. RamaNet: Computational de novo helical protein backbone design using a long short-term memory generative neural network. bioRxiv. 2020;doi:10.1101/671552.

20. Luo Y, Vo L, Ding H, Su Y, Liu Y, Qian WW, et al. Evolutionary context-integrated deep sequence modeling for protein engineering. bioRxiv. 2020;doi:10.1101/2020.01.16.908509.

21. Gligorijevic V, Renfrew PD, Kosciolek T, Leman JK, Berenberg D, Vatanen T, et al. Structure-Based Protein Function Prediction using Graph Convolutional Networks. bioRxiv. 2020;doi:10.1101/786236.

22. Strokach A, Becerra D, Corbi-Verge C, Perez-Riba A, Kim PM. Fast and flexible design of novel proteins using graph neural networks. bioRxiv. 2020;doi:10.1101/868935.

23. Sanyal S, Anishchenko I, Dagar A, Baker D, Talukdar P. ProteinGCN: Protein model quality assessment using Graph Convolutional Networks. bioRxiv. 2020;doi:10.1101/2020.04.06.028266.

24. Boyken SE, Chen Z, Groves B, Langan RA, Oberdorfer G, Ford A, et al. De novo design of protein homo-oligomers with modular hydrogen-bond network-mediated specificity. Science. 2016;352(6286):680–687.

25. Bowerman S, Wereszczynski J. Detecting Allosteric Networks Using Molecular Dynamics Simulation. Methods Enzymol. 2016;578:429–447.

26. Canutescu AA, Shelenkov AA, Dunbrack RL. A graph-theory algorithm for rapid protein side-chain prediction. Protein Sci. 2003;12(9):2001–2014.

27. Sperduti A, Starita A. Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks. 1997;8(3):714–735.

28. Gori M, Monfardini G, Scarselli F. A new model for learning in graph domains. In: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.. vol. 2. IEEE; 2005. p. 729–734.

29. Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE transactions on neural networks. 2008;20(1):61–80.

30. Qiu J, Tang J, Ma H, Dong Y, Wang K, Tang J. Deepinf: Social influence prediction with deep learning. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining; 2018. p. 2110–2119.

31. Liu Y, Shi X, Pierce L, Ren X. Characterizing and forecasting user engagement with in-app action graph: A case study of snapchat. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining; 2019. p. 2023–2031.

32. Wu Y, Lian D, Xu Y, Wu L, Chen E. Graph convolutional networks with markov random field reasoning for social spammer detection. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34; 2020. p. 1054–1061.

33. Duvenaud D, Maclaurin D, Aguilera-Iparraguirre J, Gómez-Bombarelli R, Hirzel T, Aspuru-Guzik A, et al. Convolutional networks on graphs for learning molecular fingerprints. arXiv preprint arXiv:150909292. 2015;.

34. Do K, Tran T, Venkatesh S. Graph transformation policy network for chemical reaction prediction. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining; 2019. p. 750–760.

35. Choi E, Xu Z, Li Y, Dusenberry M, Flores G, Xue E, et al. Learning the graphical structure of electronic health records with graph convolutional transformer. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34; 2020. p. 606–613.

36. Fout AM. Protein interface prediction using graph convolutional networks. Colorado State University; 2017.

37. Shlomi J, Battaglia P, Vlimant JR. Graph neural networks in particle physics. Machine Learning: Science and Technology. 2020;2(2):021001.

38. Bruna J, Zaremba W, Szlam A, LeCun Y. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:13126203. 2013;.

39. Defferrard M, Bresson X, Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems; 2016. p. 3844–3852.

40. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. International Conference of Learning Representations (ICLR). 2017;.

41. Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems; 2017. p. 1024–1034.

42. Velickovic P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y. Graph attention networks. International Conference of Learning Representations (ICLR). 2018;.

43. Xu K, Hu W, Leskovec J, Jegelka S. How Powerful are Graph Neural Networks? arXiv preprint arXiv:181000826. 2018;.

44. Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE. Neural message passing for quantum chemistry. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org; 2017. p. 1263–1272.

45. Simonovsky M, Komodakis N. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2017.

46. Simonovsky M, Komodakis N. Graphvae: Towards generation of small graphs using variational autoencoders. In: International Conference on Artificial Neural Networks. Springer; 2018. p. 412–422.

47. Xie T, Grossman JC. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. Physical review letters. 2018;120(14):145301.

48. Li G, Xiong C, Thabet A, Ghanem B. Deepergcn: All you need to train deeper gcns. arXiv preprint arXiv:200607739. 2020;.

49. Battaglia PW, Hamrick JB, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, et al. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:180601261. 2018;.

50. Prates MO, Avelar PH, Lemos H, Gori M, Lamb L. Typed graph networks. arXiv preprint arXiv:190107984. 2019;.

51. He K, Zhang X, Ren S, Sun J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification; 2015.

52. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. arXiv preprint arXiv:170603762. 2017;.

53. 8000 Filtered Structures;. `http://kinemage.biochem.duke.edu/databases/top8000.php`.

54. Maguire JB, Haddox HK, Strickland D, Halabiya SF, Coventry B, Griffin JR, et al. Perturbing the energy landscape for improved packing during computational protein design. Proteins. 2020;.

55. Grattarola D, Alippi C. Graph Neural Networks in TensorFlow and Keras with Spektral; 2020.

56. Chaudhury S, Lyskov S, Gray JJ. PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta. Bioinformatics. 2010;26(5):689–691.

57. Maguire J. MentenGCN; 2021. Available from: `https://menten-gcn.readthedocs.io/en/latest/`.

58. Chollet F, et al.. Keras; 2015. `https://keras.io`.

59. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization; 2014. Available from: `http://arxiv.org/abs/1412.6980`.

60. ResidueSelectors;. `https://www.rosettacommons.org/docs/latest/scripting_documentation/RosettaScripts/ResidueSelectors/ResidueSelectors#residueselectors_conformation-dependent-residue-selectors_layerselector`.