Efficient long-range interaction modelling in molecular graphs and proteins

Antonia Boca*
University of Cambridge
aib36@cam.ac.uk

Rishabh Jain*
University of Cambridge
rj412@cam.ac.uk

Abstract

Graph Neural Networks (GNNs) have become a popular choice for many tasks that involve dealing with relational and structural data, such as molecular function prediction. In particular, rotation- and translation-equivariant GNNs have powerful inductive biases that are needed when working with amino-acids' relative positional information within a protein. However, due to the large size of biological molecules, many models are limited to exploiting the local neighbourhood information of each amino-acid residue; this excludes the possibility of introducing information from long-range interactions, which may be crucial for determining certain properties of the molecule. This project investigates the feasibility of performing graph rewiring on molecules by generating random edges between residues in order to model *long-range interactions*. We evaluate our methods on the task of molecular function prediction and find that adding random edges does not improve the overall performance of our models. After performing both a qualitative and quantitative analysis of our dataset, we believe the reason behind this lack of improvement is that long-range interactions do not represent a performance bottleneck in molecular function prediction.

1 Introduction

Graph neural networks (GNNs) are a type of neural network that can operate on data represented in the form of graphs to perform tasks such as node and graph classification, and link prediction. Due to their highly inductive bias towards structural and relational information, GNNs have been shown to be effective for physics tasks [1–3], chemistry tasks [4, 5] and biological tasks [6, 7]. While they have also had great success in some molecular tasks such as protein-folding [8], they still struggle to achieve high performance in others, such as amino-acid residue prediction [7], where state-of-the-art models reach an accuracy of just above 50%.

One candidate explanation for the limited performance of GNNs on some molecular tasks is their inability to properly model long-range interactions between different atoms or residues, since the *message-passing* framework [9] itself focuses on aggregating local information. Depending on the downstream task, the true interaction range between the components of a molecule may be unknown; moreover, from a physics perspective, all atoms within a molecule interact with each other in some form or another. One solution to this problem is to model residues or atoms as nodes in a fully-connected graph. However, this incurs a **high computational cost**, since fully-connected interaction graphs turn into densely populated adjacency matrices for graphs that may contain even thousands of nodes.

A second solution to the aforementioned problem is to increase the number of layers within a GNN, which can increase the *receptive field* of each node in the graph. However, this approach suffers from the **over-squashing** phenomenon, first observed by Alon and Yahav [10]. Specifically, long-range GNN tasks require as many layers as the range of interaction between nodes; however, the receptive field of each node may increase exponentially with the linear increase in the number of

*Equal contribution.

A. Boca and R. Jain, Efficient long-range interaction modelling in molecular graphs and proteins. *Representation Learning on Graphs and Networks (L45), University of Cambridge*.

GNN layers, leading to information squashing from far-away nodes. In such cases, *graph rewiring* may be necessary to enable information flow between distant nodes when dealing with heterophilic networks to avoid over-squashing.

While there exist many proposals of graph rewiring techniques (see Section 1.2), in this project we focus our attention on **random edge** graph rewiring, a method in which we augment the original molecular graph with edges drawn from a random distribution, in the hope that many of these edges will successfully act as bridges between far-away parts of the molecule. This approach is inspired by Ingraham et al. [11]; since Ingraham et al. [11] themselves do not provide either theoretical or empirical justifications for their use of random-edge augmentation methods, this project strives to empirically assess the performance of the techniques proposed in their paper on the task of **Molecular Function prediction**. While final results are negative, our experiments may suggest that the performance bottleneck in many molecular tasks does not come from the inability to model long-range interactions.

1.1 Contributions

Our main contributions are the following:

- We implement a GVP-GNN model [7, 12] for the task of molecular function prediction on the Gene Ontology [13] dataset. Our baseline model matches the performance of Zhang et al. [14] who implement a similar model for the same task;
- We evaluate the performance of different edge generation methods, with the ultimate goal of performing efficient long-range interaction modelling within our models;
- We provide both a qualitative and quantitative analysis of the performance of our methods and suggest avenues for future work in the area.

1.2 Related work

Transductive graph rewiring. The literature on graph rewiring can be split into methods that perform *inductive* and *transductive* graph rewiring. The latter applies transformations to the adjacency matrix of each graph as a pre-processing step in order to improve the performance of the task at hand. Gasteiger et al. [15] introduce *diffusive rewiring* by using the Graph Diffusion Convolution (GDC) operator. This operator leverages generalised graph diffusion, a process that is also used in Google's PageRank algorithm. Another popular method for rewiring was introduced by Topping et al. [16], who describe the previously observed *over-squashing* phenomena using the *Ricci Curvature* [17], and propose a "surgical" approach to adding graph edges in order to get rid of bottlenecks.

Equivariant graph neural networks. Equivariant GNNs are a type of GNNs that are designed to be equivariant to certain types of transformation over the input data. GNNs that operate on biological or physical data are designed to be *translation* and *rotation* equivariant, with some models achieving state-of-the-art performance in tasks that benefit from using positional information [1, 7, 12, 18].

Molecular function prediction. Molecular function (MF) prediction refers to process of predicting the function of a molecule based on its structure, sequence or other attributes. In particular, molecular function is often described using terms from the **Gene Ontology** (GO) initiative [13]. The GO knowledge base contains terms for the cellular components, molecular function and biological process associated with genes and gene products, such as proteins. There exist various approaches that use machine learning approaches to predict MF that range from using sequential molecular data [19, 20] to exploiting 3D structure via GNNs [14, 21, 22], and even taking advantage of relations in the label space [23].

2 Methodology

Our project can be split into four methodological parts:

1. Process molecules into graphs that can be loaded by our code;
2. Generate edges for each of these graphs and create training, validation and testing datasets;

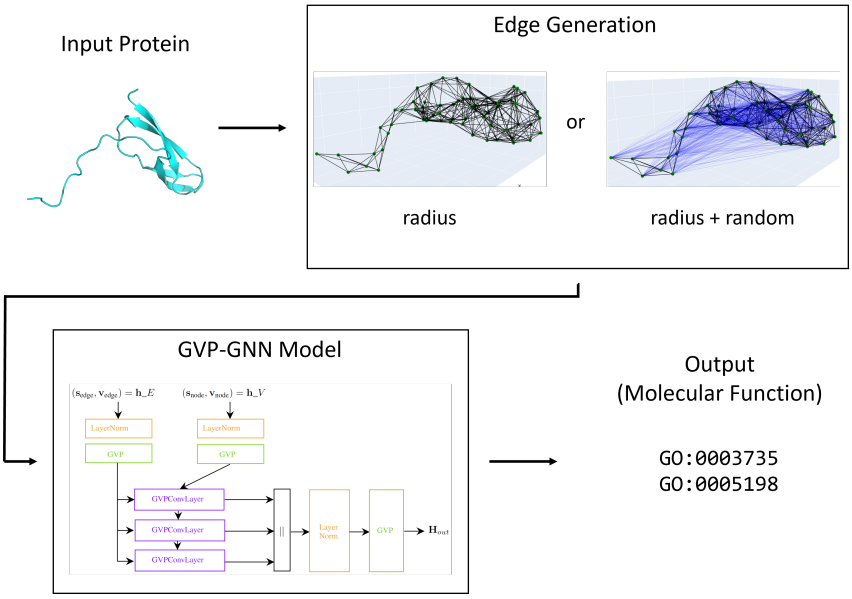


Figure 1: Overview of our data pipeline. Molecules are represented by graphs of amino-acids. We draw edges between amino-acids based on several edge generation techniques, and train GVP-GNN models using these molecular graphs on the task of molecular function prediction.

3. Train rotation- and translation-equvariant GNN models on the task of multi-label graph classification in order to predict the molecular function(s) of each molecule;
4. Analyse the performance of different edge generation methods.

We use `pytorch` [24] and `pytorch-geometric` [25] to perform data processing and design our GNN model. The dataset itself is downloaded using code inspired by `torchdrug` [26] and the training pipeline and parallelisation logic is handled through `pytorch-lightning`.[2]

The rest of this section describes the architecture of our model (Section 2.1), the formulation of molecular function prediction into a classification task (Section 2.2), and the approach to edge generation we use for long-range interaction modelling (Section 2.3). We additionally provide a summary of our data pipeline in Figure 1. An overview of the experiments and a discussion of the performance of these edge generation methods can be found in Section 3.

2.1 The GNN architecture

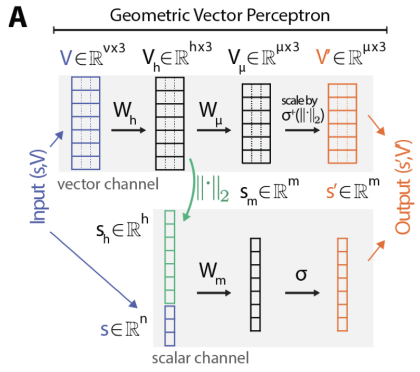
The **Geometric Vector perceptron** (GVP) is an equivariant learning module introduced by Jing et al. [12] that can be integrated within a GNN message passing layer. Its most important feature is that it can learn both scalar-valued and vector-valued functions over geometric vectors and scalars; it can be thought of as generalising the concept of a Multi-Layer Perceptron to vectors. Borrowing some notation from [12], if we have a tuple $(\mathbf{s}, \mathbf{V})$ of scalar features $\mathbf{s} \in \mathbb{R}^n$ and vector features $\mathbf{V} \in \mathbb{R}^{\nu \times 3}$, the GVP computes updated features $(\mathbf{s}', \mathbf{V}') \in \mathbb{R}^m \times \mathbb{R}^{\mu \times 3}$ as can be seen in Figure 2.

The message-passing framework. The GVP module can be incorporated in the message-passing framework to create a *GVP Convolution*. If $\mathbf{h}_{\mathcal{V}}^{(j)}$ are the node embeddings of node j; $\mathbf{h}_{\mathcal{E}}^{(j \to i)}$ are the edge embeddings of edge $(j \to i)$; $\mathbf{h}_m^{(j \to i)}$ is the message passed from node j to node i; and k' is the number of incoming messages, then we can define the module:

$$\mathbf{h}_m^{(j \to i)} := g_1\Big(\text{concat}(\mathbf{h}_{\mathcal{V}}^{(j)},\ \mathbf{h}_{\mathcal{E}}^{(j \to i)})\Big) \tag{1}$$

$$\mathbf{h}_{\mathcal{V}}^{(i)} \leftarrow \text{LayerNorm}\Big(\mathbf{h}_{\mathcal{V}}^{(i)} + \frac{1}{k'}\text{Dropout}\Big(\sum_{j:\mathbf{e}_{j \to i} \in \mathcal{E}} \mathbf{h}_m^{(j \to i)}\Big)\Big) \tag{2}$$

[2] https://lightning.ai/docs/pytorch/stable/



Input: Scalar and vector feats. $(\mathbf{s}, \mathbf{V}) \in \mathbb{R}^n \times \mathbf{R}^{\nu \times 3}$
Output: Scalar and vector feats. $(\mathbf{s}', \mathbf{V}') \in \mathbb{R}^n \times \mathbb{R}^{\mu \times 3}$
$h \leftarrow \max(\nu, \mu)$
$V_h \leftarrow \mathbf{W}_h \mathbf{V} \in \mathbb{R}^{h \times 3}$
$V_\mu \leftarrow \mathbf{W}_\mu \mathbf{V}_h \in \mathbb{R}^{\mu \times 3}$
$\mathbf{s}_h \leftarrow ||V_h||_2$ (row-wise) $\in \mathbb{R}^h$
$\mathbf{v}_\mu \leftarrow ||V_\mu||_2$ (row-wise) $\in \mathbb{R}^\mu$
$\mathbf{s}_{h+n} \leftarrow \text{concat}(\mathbf{s}_h, \mathbf{s}) \in \mathbb{R}^{h+n}$
$\mathbf{s}_m \leftarrow \mathbf{W}_m \mathbf{s}_{h+n} + \mathbf{b} \in \mathbb{R}^m$
$\mathbf{s}' \leftarrow \sigma(\mathbf{s}_m) \in \mathbb{R}^m$
$\mathbf{V}' \leftarrow \sigma^+(\mathbf{v}_\mu) \circledcirc \mathbf{V}_\mu$ (row-wise multiplication) $\in \mathbb{R}^{\mu \times 3}$
return $(\mathbf{s}', \mathbf{V}')$

Figure 2: *(Left)* An illustration of the data flow of the module; *(Right)* Algorithm showing the computation involved in a GVP module. Both the pseudocode and the illustration are taken from [12].

We follow this module with a *feed-forward point-wise* layer to update the node embeddings:

$$\mathbf{h}_{\mathcal{V}}^{(i)} \leftarrow \text{LayerNorm}\Big(\mathbf{h}_{\mathcal{V}}^{(i)} + \text{Dropout}\big(g_2(\mathbf{h}_{\mathcal{V}}^{(i)})\big)\Big) \tag{3}$$

Functions g_1 and g_2 are compositions of three and two GVPs, respectively. Therefore, a **GVP Convolutional Layer** consists of one GVP Convolution and one feed-forward point-wise update layer. The final architecture is composed of GVP modules, layer normalisation modules and GVP Convolutional layers, as can be seen in Figure 3.

Graph readout. Our goal is to predict a set of molecular functions for each molecule in the dataset, so we formulate the problem as a graph classification task. We denote the matrix that contains the final features for each node in a graph as $\mathbf{H}_{out}$ and obtain the final graph representation for each molecule by summing together the final features of all nodes:

$$\mathbf{h}^{\text{readout}} = \sum_{i=1}^{N} [\mathbf{H}_{out}]_i \tag{4}$$

$\mathbf{h}^{\text{readout}}$ is then passed through an MLP in order to get the final class probabilities for every possible molecular function, as will be explained in more detail in Section 2.2.

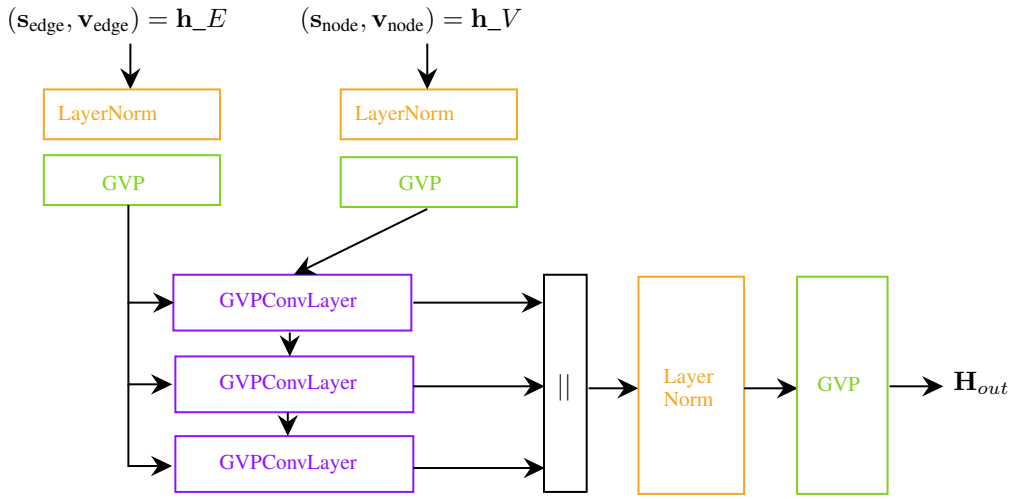


Figure 3: Diagram of a GVP-GNN with 3 GVP Convolutional Layers. The outputs of each convolutional layer are concatenated before being passed through the `LayerNorm`.

Node and edge features. Molecules are composed of amino-acids chained together. In this project, we model molecules as **graphs** composed of **nodes** representing amino-acids. In order to take advantage of structural information, we generate node features that indicate the position and orientation of each amino-acid in space, following Jing et al. [12].

Each amino-acid consists of 4 *backbone* atoms (C_α[3], C, N and T). To represent molecular structure using scalar and vector features, consider X_i to be the position of atom X in the i-th amino-acid of a molecule. Then, the node features we extract are the following:

- 6 scalar features representing $\sin\phi$ and $\cos\phi$, with ϕ being each of the three dihedral angles computed from C_{i-1}, N_i, C_{α_i}, C_i and N_{i+1};
- Two vectors indicating the *orientation* of the amino-acid. These vectors are the *forward* vector $C_{\alpha_{i+1}} - C_{\alpha_i}$ and the *reverse* vector $C_{\alpha_{i-1}} - C_{\alpha_i}$.
- The unit vector $\mathbf{v}_i$ representing the imputed direction of $C_{\beta_i} - C_{\alpha_i}$[4], which can be computed by noting $\mathbf{n} = N_i - C_{\alpha_i}$ and $\mathbf{c} = C_i - C_{\alpha_i}$ in the following equation:

$$\mathbf{v}_i = \sqrt{\frac{1}{3}}\frac{\mathbf{n}\times\mathbf{c}}{||\mathbf{n}\times\mathbf{c}||_2} - \sqrt{\frac{2}{3}}\frac{\mathbf{n}+\mathbf{c}}{||\mathbf{n}+\mathbf{c}||_2} \tag{5}$$

- The one-hot representation of the amino-acid identity.

Similarly, edge $(j \to i)$ has embedding $\mathbf{h}_e^{(j\to i)}$ containing:

- The unit vector in the direction of $C_{\alpha_j} - C_{\alpha_i}$;
- The encoding of the distance $||C_{\alpha_j} - C_{\alpha_i}||_2$ in 16 Gaussian radial basis functions;
- A sinusoidal encoding of $j - i$, representing the distance along the backbone.

2.2 Task details

The downstream task of this project is to predict **molecular function** by employing a GVP-GNN to take advantage of the structural information of the molecule. Molecular function is defined as a molecular process that can be carried out by the action of a single macromolecular machine such as a protein. These functions have standardised terms[5] proposed by the Gene Ontology initiative [13]. A molecule can have multiple functions and therefore can be assigned multiple *GO* terms. Figure 4 shows the cartoon of a molecule with over 700 atoms that has 5 molecular functions. We use the `GeneOntology` molecule dataset provided by the `TorchDrug` library [26]. It consists of a split of 27,496 train samples, 3,053 validation samples and 2,991 test samples; this split is also used in Gligorijevic et al. [21].

Multi-label classification task. Notice how the terms in Figure 4 are related; in fact, rather than representing independent categories, many functions are correlated, and even hierarchical. For example, `GO:0019842` is a more general version of `GO:0030170`. Spalević et al. [23] model the *label space* relations using so-called "Tail-GNNs" in order to properly account for the correlations between different terms.

Due to the time constraints imposed on this project, we decide to model the MF prediction task as a *multi-label classification task*, by taking inspiration from Gligorijevic et al. [21]. To account for the fact that some terms are more general than others (and therefore will have higher frequency) we compute class weights using the training set. Overall, **the objective** of our task is to minimise the following weighted binary cross-entropy cost function:

$$\mathcal{L}(\Theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|GO|} w_j y_{ij}\log(\hat{y}_{ij}) \tag{6}$$

where Θ are the set of all parameters in the model; $w_j = \frac{N}{N_j^+}$ is the class weight for function j, with N_j^+ being the number of positive examples associated with class j; N is the total number of

[3] C_α is the central carbon atom of the amino-acid.

[4] C_β represents the second carbon from the carboxyl atom C.

[5] There are 489 standard MF terms in the Gene Ontology knowledge base.

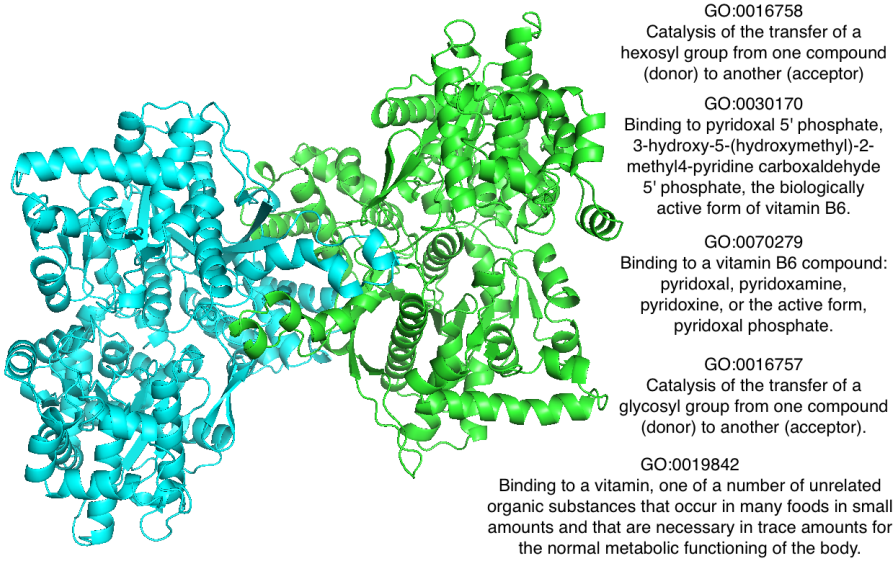


Figure 4: Molecule introduced in O'Reilly et al. [27] and its associated molecular function GO terms [13]. The molecule cartoon was generated using PyMol [28].

samples; and $|GO|$ is the total number of classes (i.e., GO terms). y_{ij} is the true binary indicator for whether sample i has label j and $\hat{y}_{ij}$ is the predicted probability that sample i is annotated with class j. As mentioned in Section 2.1, $\mathbf{h}_i^{\text{readout}}$ from Equation 4 is passed through an MLP to predict the probability that molecule i has molecular function j by returning prediction vector $\hat{\mathbf{y}}_i \in \mathbb{R}^{|GO|}$.

Evaluation metrics. To evaluate the performance of our trained models on the GO dataset, we use the maximum F-score $\text{F}_{\max}$ that is calculated by computing the average precision(t) and recall(t) for a given decision threshold $t \in [0, 1]$ from the entire dataset. Formally, the maximum F-score is:

$$\text{F}_{\max} = \max_t \left\{ \frac{2 \cdot \text{precision}(t) \cdot \text{recall}(t)}{\text{precision}(t) + \text{recall}(t)} \right\} \tag{7}$$

A more detailed explanation of how the maximum F-score is computed can be found in Appendix A.

2.3 Edge generation methods

Graphs with sparse edges, such as edges that are present only between nodes that have a chemical bond between them, tend to perform worse than dense graphs which have full connectivity when reasoning over the global structure and properties of the molecule [29, 30]. However, dense graphs bring about quadratic scaling costs, so local graphs that use k-Nearest Neighbours (k-NN) [31] or radius cutoffs [32] for edge generation are usually used. Ingraham et al. [11] propose edge generation methods aimed at adding long-range reasoning to local graphs while preserving sub-quadratic scaling.

Chroma. Chroma [11] is a generative model for proteins and protein complexes. It introduces random graph neural networks that enable long-range reasoning with sub-quadratic scaling. These graphs have similar connectivity statistics to efficient N-body physics simulations, like the Barnes-Hut simulation [33].

Many N-body physics simulations (like gravitational simulation) involve $\mathcal{O}(N^2)$ dense interactions. Methods like Barnes-Hut allow us to effectively simulate these with only $\mathcal{O}(N \log N)$-scaling computation by modelling distant interactions more coarsely. Drawing inspiration from these, Ingraham et. al. [11] propose (mixed) random graph generation for Chroma, where instead of having fully connected graphs, we have two types of edges: *local edges* and *random edges*. The local edges allow for fine-grained modelling of local effects, while the random edges add long-range context. Chroma adds a fixed number k local edges and k_{random} random edges. The local edges are added

using k-NN, while the random edges are sampled from a random edge propensity distribution. The sampling is done using the Gumbel top-k trick, a generalisation of Gumbel max-trick.

Gumbel Max-Trick. Suppose that we want to sample a discrete random variable I which has a categorical distribution Categorical$(p_1, \ldots, p_n)$, i.e. I has domain $N = \{1, \ldots, n\}$ with $P(I = i) = p_i \;\; \forall i \in N$. Further, suppose that we have access to the un-normalized log-probabilities ϕ_i, i.e. $\exp(\phi_i) \propto p_i$ for $i \in N$. Gumbel Max-Trick [34] allows us to sample from the categorical distribution by sampling variables from the Gumbel distribution and taking the max of these.

A variable $G \sim$ Gumbel(ϕ) is said to have Gumbel distribution [35] with location ϕ, if $G = \phi - \log(-\log U)$, where $U \sim$ Uniform$(0, 1)$. Gumbel max-trick uses the fact that if we have some random variables $G_i \sim$ Gumbel(ϕ_i), $i \in N$, and let $I = \text{argmax}_i(G_i)$, then $I \sim$ Categorical$(p_1, \ldots, p_n)$.

Gumbel Top-k Trick. The Gumbel Max-trick can be generalized to the Gumbel Top-k trick which aims to draw an ordered sample of size k without replacement. This is done by generalising 'argmax' with 'argtop-k', which given a sequence of values, returns the indices of the highest k values in a descending order.

Random edge distributions. We run three experiments to assess the performance of three edge generation distributions: uniform, exponential and inverse cubic. Table 1 shows the results of each of the three distributions on a GVP-GNN model with 3 layers. For each graph we generate 20 random edges and all local edges within a radius of 5Å.

Table 1: Results of experiments with various edge generation distributions. We find that the exponential and the inverse cubic distribution perform similarly.

Random distribution	Edge propensity formula $p(\mathcal{E}_{ij} \in \mathcal{G} \vert D_{ij}(x)) \propto$	F_{max} score	Test loss
Uniform	Constant	0.3641	0.0020
Exponential	$\exp(-D_{ij}(x)/I)$	0.3992	0.0026
Inverse cubic	$1/D_{ij}(x)^3$	0.3945	0.0028

We choose the **inverse-cubic edge propensity** for the rest of our experiments, i.e. $p(\mathcal{E}_{ij} \in \mathcal{G}(\mathbf{x}) \vert D_{ij}(\mathbf{x})) = D_{ij}(\mathbf{x})^{-3}$. This propensity is used by Ingraham et al. [11] because it leads to connectivity scaling similar to Barnes-Hut [33]. For more details, refer to Appendix B.

3 Experiments

Chroma [11] introduce random edge generation for efficient long-range interaction modelling with GNNs but they do not provide experimental results or public code for these. In our experiments, we explore two potential uses for random edge generation:

- improving performance of locally connected models by adding long-range context. We hypothesize that the model trained using graphs with randomly generated edges would perform **better** than a baseline model trained on graphs with local edges.
- modelling interactions more efficiently. We hypothesise that a *shallower* model trained using graphs with random edges can **match** the performance of a *deeper* model trained using graphs with local edges.

We consider three different random edge generation methods:

1. the *radius* method. This method draws an edge between every pair of amino-acid residues whose C_α carbons are within a specified distance;
2. the *knn* method. This method draws edges to the $k = 20$ nearest neighbouring amino-acids to every amino-acid;

3. the *random* method. This method draws edges from each amino-acid to every other amino-acid according to a random distribution; here we use the inverse cubic-distance propensity, as detailed in Section 2.3.

Note that the first and second edge generation methods are *local* to the neighbourhood of each node. We evaluate the performance of each of the three methods first individually and then in combination.

More details on the hyperparameters of our experiments can be found in Appendix C.

3.1 Use 1: improving models with long-range interactions

Table 2 shows the evaluation results for different combinations of methods and edge parameters. Note that the best performing model is also the **baseline** model, which follows the hyperparameters used in Zhang et al. [14]. We also note that the baseline model has a similar performance to the one reported in [14], suggesting our choice of hyperparameters is close to optimal for the task at hand.

We find that all methods have similar performance, with the exception of the method using a radius cutoff of 5Å. The random edge generation methods do not outperform the baseline method with a radius cutoff of 10Å.

Table 2: Evaluation results for 3-layer GVP-GNN models using different edge generation methods. Each statistic is generated from 3 runs with 3 different seeds. For the F_{max} score we report both the mean and the standard deviation. The best performing edge method is highlighted in gray.

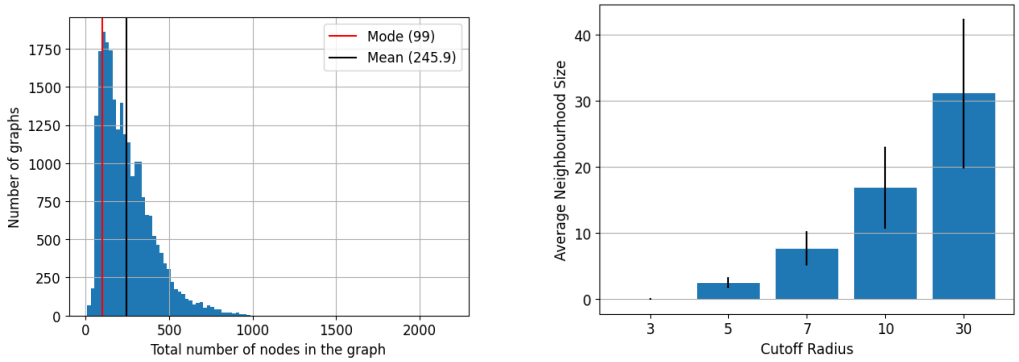
Edge generation method	Radius size (Å)	Number of random edges per node	F_{max} score	mean loss	Average running time (hh:mm:ss)
radius	5	-	$0.3452_{\pm 0.0059}$	0.00304	06:53:50
radius	10	-	$0.4126_{\pm 0.0030}$	0.00282	06:42:54
random	-	40	$0.3996_{\pm 0.0045}$	0.00235	13:47:02
knn (20)	-	-	$0.4095_{\pm 0.0045}$	0.00293	07:18:06
knn (20) + random	-	40	$0.4038_{\pm 0.0045}$	0.00256	15:43:46
radius + random	5	40	$0.4000_{\pm 0.0030}$	0.00244	15:47:19
radius + random	10	40	$0.4088_{\pm 0.0031}$	0.00250	14:47:16

Discussion. We perform a quantitative analysis of the input dataset, as shown in Figure 5. Figure 5(a) shows the distribution of graph sizes in the training dataset and Figure 5(b) shows the average neighbourhood size of each node in the training dataset at different radius cutoffs.

We find two leading explanations for the similar performance of most of our models:

1. **The receptive field of each node contains the entire molecule**. By looking at Figure 5, we find that a radius cutoff of 10Å provides each node with a neighbourhood size of almost 20 edges; given that the GVP-GNN model has 3 message-passing layers and that the median graph has 99 nodes, we believe that the *receptive field* of each node can potentially include **every other node** in the graph. This means that adding new (potentially random) edges does not provide the model with any new long-range interactions. This explains why all models (with the exception of the radius method using a cutoff of 5Å) perform similarly, since having around 16-20 edges already guarantees that each node will have access to all other nodes.
2. **Long-range interactions are not as important for function prediction**. A counterargument to explanation 1 is that even when nodes can perceive the entire graph through their receptive field, *over-squashing* [10] will prohibit the model from extracting enough valuable information on far-away edges. However, we argue that the task at hand does not benefit from more long-range information than it already receives by using a radius cutoff of 10Å. In particular, we notice how even using a radius cutoff of 5Å, which effectively means each node gets 2-3 edges, the model still reaches a surprising F_{max} score of **0.34** on average.
   Furthermore, the bottom row of Figure 6 shows a series of graphs with high loss in the test set. Even when the graphs consist of long chains of nodes, the random edges ensure that the molecule stays well-connected, and so it is unlikely that node connectivity is the issue in these graphs.

Given these two arguments, *we believe that this performance ceiling does not come from the inability to model long-range interactions*, and is more likely to do with the model architecture and its ability to handle structural information. We discuss avenues for future work in this context in Section 4.



(a) Graph size distribution for the training dataset.

(b) Average neighbourhood sizes for different radius cutoffs, with one standard deviation error bars.

Figure 5: Dataset analysis plots.

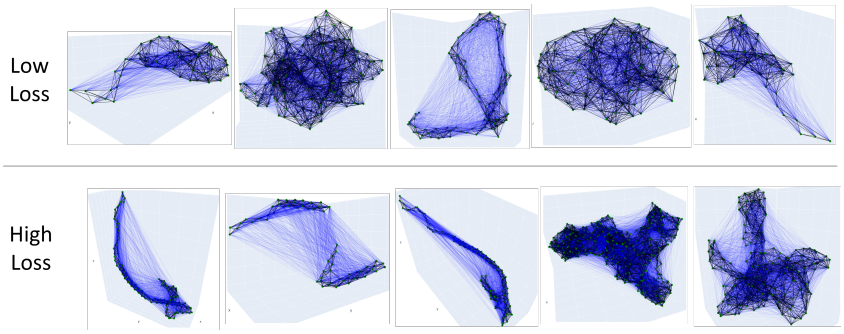


Figure 6: Graphs with *radius + random* edge generation. The top row consists of 5 graphs with low loss in the test set. The bottom row consists of 5 graphs with high loss in the test set. Notice how all graphs are well connected.

3.2 Use 2: modelling interactions more efficiently

One of the main insights gained from running experiments in Section 3.1 is that we can model all necessary interactions to predict molecular function by using local edges with a 3-layer model. The choice of using an architecture with 3 layers follows a practitioner's intuition that can be found in other works using GVP models [7, 12, 14]. However, using so many layers incurs a computational cost that we believe can be ameliorated by using random edges, as this allows us to **maintain** the size of each node's receptive field while **reducing** the number of layers in our model.

We reduce the number of layers in our GVP-GNN architecture to 2 and run experiments with several edge generation methods in order to test our hypothesis. Table 3 summarises the results of all experiments run with a 2-layer GVP-GNN.

Discussion. The results of our second round of experiments echo the explanations we provided in Section 3.1. We notice how the results of GVP-GNN models with 2 layers are similar to the results of models with 3 layers. For example, using a radius cutoff of 5Å or 10Å gives effectively the *same* performance, regardless of the number of layers used in the model. This again suggests that *long-range interactions are not the bottleneck*, and having a smaller receptive field is not the main problem in solving these types of molecular tasks. Figure 7 shows how some graphs have low loss (top row) or high loss (high row) regardless of the type of edges we generate.

Table 3: Evaluation results for 2-layer GVP-GNN models using different edge generation methods. Each statistic is generated from 3 runs with 3 different seeds. For the F_{max} score we report both the mean and the standard deviation. The best performing edge methods are highlighted in gray.

Edge generation method	Radius (Å)	Number of random edges per node	Fmax Score	Mean loss	Average runtime (hh:mm:ss)
radius	3	-	$0.2749_{\pm 0.0086}$	0.00186	06:25:39
radius	5	-	$0.3480_{\pm 0.0110}$	0.00296	06:32:12
radius	10	-	$0.4106_{\pm 0.0055}$	0.00278	06:14:51
random	-	20	$0.3863_{\pm 0.0036}$	0.00213	07:30:17
radius + random	3	20	$0.4043_{\pm 0.0050}$	0.00276	07:34:14
radius + random	3	40	$0.4041_{\pm 0.0070}$	0.00261	09:41:46
radius + random	5	20	$0.3968_{\pm 0.0051}$	0.00262	07:30:02

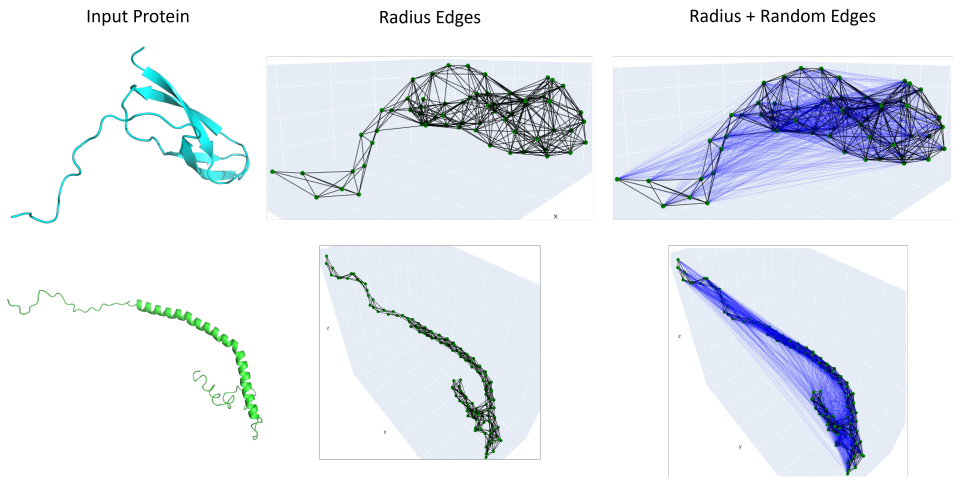


Figure 7: Graphs with edges generated using a radius cutoff or a mixed method. The top row illustrates a molecule that has a low loss regardless of the edge method used. Similarly, the bottom row illustrates a molecule that has a high loss regardless of the number of long-range interactions it can model.

4 Conclusion

In this project, we investigated the performance of using several edge generation methods to efficiently model long-range interactions in molecules. We empirically assessed the performance of these edge methods by training a GVP model [12] on the task of molecular function prediction. We have found that adding long-range interactions *does not improve* the performance of our models. Furthermore, our analysis suggests that the reason for this lack of improvement is that long-range interactions do not present a major performance bottleneck and are already efficiently integrated by using local edges. We would further like to address some limitations of our work and avenues for future investigations.

Are they really *long-range* interactions? As discussed in Section 3.1, the median molecule in our dataset has 99 amino-acids, and a radius cutoff of 10Å gives each amino-acid between 15 and 20 edges. This means that in many cases, the receptive field of a node in a GVP-GNN model with 3 layers already contains the entire molecule. Therefore, we can argue that most of the molecules in the GO dataset are small enough to not have long-range interactions at all. Future work should focus on investigating the use of random edge generation methods in tasks that involve much bigger proteins, such as the ones presented in the Chroma paper [11].

How good is the GVP model anyway? In this project, we trained a GVP-GNN model [7, 12]. As we have seen in Section 3.2, the bottleneck in our case does not come from the lack of long-range

interaction modelling, but rather from the limited ability of the GVP model to handle all molecular information. However, edge generation methods may benefit other architectures, such as GCN [36] used in DeepFRI [21] for MF prediction. Therefore, more types of models should be trained with random edge methods in order to assess the overall performance of molecular graph rewiring.

Is structure all we need? Lastly, we would like to point out that this project has used a purely structural approach to solve the task of molecular function prediction. However, recent approaches to many molecular tasks involve a combination of many models, each with a different strength. For example, Zhang et al. [14] combine a language model (to extract sequence information) and a GVP model (to extract structural information) and use a contrastive learning approach in order to reach state-of-the-art performance on several molecular tasks. Given such a wide range of eclectic approaches to protein prediction, future work could focus on understanding the *importance* of modelling long-range predictions in the first place; then we may finally understand how important it is to have the right range to start with.

Statement of contribution

We, Antonia Boca and Rishabh Jain of the University of Cambridge, jointly declare that our work towards this project has been executed as follows:

- **Antonia Boca** contributed by devising and implementing the data pipeline for GO molecules, the baseline GVP-GNN model, and the training loop; running all experiments; and writing the accompanying sections in the report.
- **Rishabh Jain** contributed by implementing the edge generation methods; (co-)devising the experiments; analysing the dataset and produced graphs, generating the relevant figures; and writing the relevant sections in the report.

We both independently submit identical copies of this paper, certifying this statement to be correct.

GitHub repository with commit log

The companion source code for our project may be found at: `https://github.com/semiluna/L45`.

References

[1] Ilyes Batatia, Dávid Péter Kovács, Gregor NC Simm, Christoph Ortner, and Gábor Csányi. Mace: Higher order equivariant message passing neural networks for fast and accurate force fields. *arXiv preprint arXiv:2206.07697*, 2022. 1, 2

[2] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. *Advances in neural information processing systems*, 30, 2017.

[3] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International conference on machine learning*, pages 2688–2697. PMLR, 2018. 1

[4] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608, Aug 2016. ISSN 1573-4951. doi: 10.1007/s10822-016-9938-8. URL https://doi.org/10.1007/s10822-016-9938-8. 1

[5] Kien Do, Truyen Tran, and Svetha Venkatesh. Graph transformation policy network for chemical reaction prediction. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 750–760, 2019. 1

[6] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. *Advances in neural information processing systems*, 30, 2017. 1

[7] Bowen Jing, Stephan Eismann, Pratham N Soni, and Ron O Dror. Equivariant graph neural networks for 3d macromolecular structure. *arXiv preprint arXiv:2106.03843*, 2021. 1, 2, 9, 10

[8] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Kathryn Tunyasuvunakool, Olaf Ronneberger, Russ Bates, Augustin Žídek, Alex Bridgland, et al. Alphafold 2. *In Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book*, 2020. 1

[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 1

[10] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020. 1, 8

[11] John Ingraham, Max Baranov, Zak Costello, Vincent Frappier, Ahmed Ismail, Shan Tie, Wujie Wang, Vincent Xue, Fritz Obermeyer, Andrew Beam, and Gevorg Grigoryan. Illuminating protein space with a programmable generative model. *bioRxiv*, 2022. doi: 10.1101/2022.12.01.518682. URL https://www.biorxiv.org/content/early/2022/12/02/2022.12.01.518682. 2, 6, 7, 10, 15

[12] Bowen Jing, Stephan Eismann, Patricia Suriana, Raphael JL Townshend, and Ron Dror. Learning from protein structure with geometric vector perceptrons. *arXiv preprint arXiv:2009.01411*, 2020. 2, 3, 4, 5, 9, 10

[13] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Angela P Davis, Karen Dolinski, Selina S Dwight, Janan T Eppig, Midori A Harris, David P Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C Matese, Jane E Richardson, Martin Ringwald, Gerald M Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nature genetics*, 25(1):25–29, 2000. 2, 5, 6

[14] Zuobai Zhang, Minghao Xu, Arian Jamasb, Vijil Chenthamarakshan, Aurelie Lozano, Payel Das, and Jian Tang. Protein representation learning by geometric structure pretraining, 2023. 2, 8, 9, 11, 15

[15] Johannes Gasteiger, Stefan Weiß enberger, and Stephan Günnemann. Diffusion improves graph learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/23c894276a2c5a16470e6a31f4618d73-Paper.pdf. 2

[16] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature, 2022. 2

[17] Richard S. Hamilton. The ricci flow on surfaces. *Contemporary Mathematics*, 71:237–261, 1988. doi: 10.1090/conm/071/954419. 2

[18] Víctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9323–9332. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/satorras21a.html. 2

[19] Maxat Kulmanov, Mohammed Asif Khan, and Robert Hoehndorf. Deepgo: predicting protein functions from sequence and interactions using a deep ontology-aware classifier. *Bioinformatics*, 34(4):660–668, 2018. 2

[20] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, 2020. doi: 10.1101/622803. URL https://www.biorxiv.org/content/early/2020/12/15/622803. 2

[21] Vladimir Gligorijevic, P. Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Kyunghyun Cho, Tommi Vatanen, Daniel Berenberg, Bryn Taylor, Ian M. Fisk, Ramnik J. Xavier, Rob Knight, and Richard Bonneau. Structure-based function prediction using graph convolutional networks. *bioRxiv*, 2019. doi: 10.1101/786236. URL https://www.biorxiv.org/content/early/2019/10/04/786236. 2, 5, 11

[22] Zuobai Zhang, Minghao Xu, Vijil Chenthamarakshan, Aurélie Lozano, Payel Das, and Jian Tang. Enhancing protein language models with structure-based encoder and pre-training, 2023. 2

[23] Stefan Spalević, Petar Veličković, Jovana Kovačević, and Mladen Nikolić. Hierarchical protein function prediction with tail-gnns, 2020. 2, 5

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. 3

[25] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019. 3

[26] Zhaocheng Zhu, Chence Shi, Zuobai Zhang, Shengchao Liu, Minghao Xu, Xinyu Yuan, Yangtian Zhang, Junkun Chen, Huiyu Cai, Jiarui Lu, et al. Torchdrug: A powerful and flexible machine learning platform for drug discovery. *arXiv preprint arXiv:2202.08320*, 2022. 3, 5

[27] Marc O'Reilly, Kim A. Watson, Reinhard Schinzel, Dieter Palm, and Louise N. Johnson. Oligosaccharide substrate binding in *Escherichia coli* maltodextrin phosphorylase. *Nature Structural Biology*, 4(5):405–412, 1997. ISSN 1545-9985. doi: 10.1038/nsb0597-405. URL https://doi.org/10.1038/nsb0597-405. 6

[28] LLC Schrödinger and Warren DeLano. Pymol. URL http://www.pymol.org/pymol. 6

[29] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. 6

[30] Namrata Anand and Tudor Achim. Protein structure and sequence generation with equivariant denoising diffusion probabilistic models, 2022. 6

[31] John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. Generative models for graph-based protein design. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc,

E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/f3a4ff4839c56a5f460c88cce3666a2b-Paper.pdf. 6

[32] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P. Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E. Smidt, and Boris Kozinsky. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature Communications*, 13 (1), may 2022. doi: 10.1038/s41467-022-29939-5. URL https://doi.org/10.1038%2Fs41467-022-29939-5. 6

[33] J. H. Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324: 446–449, 1986. 6, 7

[34] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/309fee4e541e51de2e41f21bebb342aa-Paper.pdf. 7

[35] Emil Julius Gumbel. *Statistical Theory of Extreme Values and Some Practical Applications*. 1 1954. 7

[36] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 11

[37] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 16

A Evaluation metrics

To evaluate the performance of our trained models on the GO dataset, we use the maximum F-score $\mathrm{F}_{\max}$ that is calculated by computing precision and recall for a given decision threshold $t \in [0, 1]$ for each molecule and then taking the average score over all molecules. Formally, given a target molecule i and a decision threshold $t \in [0, 1]$, precision and recall are:

$$\text{precision}_i(t) = \frac{\sum_f \mathbb{1}[f \in P_i(t) \cap T_i]}{\sum_f \mathbb{1}[f \in P_i(t)]} \tag{8}$$

$$\text{recall}_i(t) = \frac{\sum_f \mathbb{1}[f \in P_i(t) \cap T_i]}{\sum_f \mathbb{1}[f \in T_i]} \tag{9}$$

where f is a function term in the ontology, T_i is a set of experimentally determined function terms for protein i, $P_i(t)$ denotes the set of predicted terms for protein i with scores greater than or equal to t and $\mathbb{1}[\cdot] \in \{0, 1\}$ is an indicator function. The average precision and recall over all proteins at threshold t is defined as:

$$\text{precision}(t) = \frac{1}{M(t)} \sum_i \text{precision}_i(t) \tag{10}$$

$$\text{recall}(t) = \frac{1}{N} \text{recall}_i(t) \tag{11}$$

where N is the total number of molecules and $M(t)$ is the number of molecules on which at least one prediction was made above the threshold t. Then, the $\mathrm{F}_{\max}$ score is defined as:

$$\mathrm{F}_{\max} = \max_t \left\{ \frac{2 \cdot \text{precision}(t) \cdot \text{recall}(t)}{\text{precision}(t) + \text{recall}(t)} \right\} \tag{12}$$

B Inverse-Cubic Edge Propensity

Let us suppose that we have a matrix $\mathbf{x}$ of co-ordinates of the nodes of input molecule. We wish to generate a graph $\mathcal{G}(\mathbf{x})$ from this, by sampling edges from some distribution. Let $\mathcal{E}_{ij}$ be an edge between nodes i and j, with distance between the nodes as $D_{ij}(\mathbf{x})$. Thus, we can write the edge propensity as $p(\mathcal{E}_{ij} \in \mathcal{G}(\mathbf{x}) | D_{ij}(\mathbf{x}))$.

When we use inverse-cubic edge propensity, we have $p(\mathcal{E}_{ij} \in \mathcal{G}(\mathbf{x}) | D_{ij}(\mathbf{x})) = D_{ij}(\mathbf{x})^{-3}$. This means that the cumulative edge propensity scales as $\int_{D_{\min}}^{D_{\max}} \frac{1}{r^3} r^2 \, dr = \log D_{\max} - \log D_{\min}$. This means that as the total radius of the system increases by $D_{\max}$, we only need to increase the total number of edges per node by a factor of $\log D_{\max}$ to keep up with the increase in total edge propensity, so that the newer edges don't stealing the edge mass from closer nodes of the graph. This means that the total computation required for a system of N atoms will scale as $\mathcal{O}(N \log N)$, which matches the scaling of Barnes-Hut algorithm. In practice, Ingraham et. al. use mixed graphs with a constant number of random edges [11]. We also follow suit.

C Experiment details

We run all experiments three times with the following three seeds: `{42, 666013, 76}` and report the average statistics. All experiments are run using 2 NVIDIA A100 GPUs on the Cambridge HPC Cluster.

Data parallalelisation is handled by `pytorch-lightning` by specifying `nodes=1`, `gpus=2` and `strategy=ddp` to the Pytorch-Lightning `Trainer`.

C.1 Hyperparameters

We take inspiration from the hyperparameters used by Zhang et al. [14] to train their GVP model for MF prediction. Table 4 summarises both the training and architecture parameters that we use.

Table 4: Model and training hyperparameters.

Hyperparameter	Value
`GVPConvLayers`	3
Node feature dimension $(\mathbf{s}, \mathbf{V})$	(100, 32)
Edge feature dimension $(\mathbf{s}, \mathbf{V})$	(64, 1)
Dropout probability	0.1
Batch size	64
Optimiser	Adam [37]
Learning rate	5e-4
Epochs	200

C.2 Edge methods

When running experiments with GVP-GNN models that contain 2 layers, we find that when combining two edge methods, they perform better if the two sets of generated edges do not overlap (this can be specified using the command line parameter `--edge_params combiner aggregation mask_prev` when running an experiment.

We also find that setting the inverse temperature equal to 10 when generating random edges based on their inverse cubic-distance propensity performs better, so we run all our experiments with the command line parameter `--edge_params random inverse_temp 10`.