A JULIA IMPLEMENTATION OF:

Hierarchical Graph Transformer for Protein Function Prediction

M. Mahlum
University of Minnesota – Twin Cities
monte.mahlum@mail.mcgill.ca

Abstract

HEALJ is a Julia reimplementation of the HEAL architecture for protein function prediction from ESM-1b sequence embeddings and PDB-derived protein graphs. We define and implement the full model, including a GCN encoder and hierarchical graph transformer, with a symmetric InfoNCE contrastive loss. On a subset of 28,289 training proteins (3,135 validation, 3,123 test), HEALJ trains reasonably on a single NVIDIA GeForce RTX 5060 and reached $F_{\max} = 0.457$ at $t = 0.04$ after 25 epochs, providing a reproducing reference for HEAL and an extensible basis for further graph-based protein function models in Julia.

Repository: github.com/monte-mahlum/HEALJ

Acknowledgments

The author would like to first acknowledge those responsible for introducing and originally implementing the HEAL model: (Gu et al. 2023). Additionally, we would like to thank Professor Dan Boley for his support and guidance throughout this rich course.

1. Introduction

1.1 Biological Preliminaries

1.1.1 Proteins, Amino Acids, and Residues

The primary structure of a protein is given by its defining amino acid sequence (Figure 1**a**). After binding in the sequence, each amino acid corresponds to a unique residue which can be thought of as that which is left over after the binding. These residues, which are bonded to the C_α (alpha carbon) atom in the amino acid, have the disposition to interact with (1) each other, forming the secondary and tertiary structures (Figure 1**b**); (2) other proteins, forming quaternary structure; and (4) the environment, realizing complex capabilities of the protein, e.g., catalytic behavior.

1.1.2 Gene Ontology

Introduced in the year 2000, the Gene Ontology (GO) aims to develop a standardized vocabulary to describe what biological entities *are* and *do*. At the time of this writing, there are 39,354 standardized GO terms with thousands more being proposed each year (Gene Ontology Consortium 2025). At its core, the GO terms carry an is a-hierarchy forming a directed acyclic graph as each term may inherit multiple parents (see Figure 2). GO terms are further connected through their ontological definitions, e.g., as seen in Figure 2, the external encapsulating structure is part of the cell periphery.

This ontology enables proteins, among other biological entities, to be characterized by those GO terms to which they are related. While GO may be incredibly expressive, such a large vocabulary

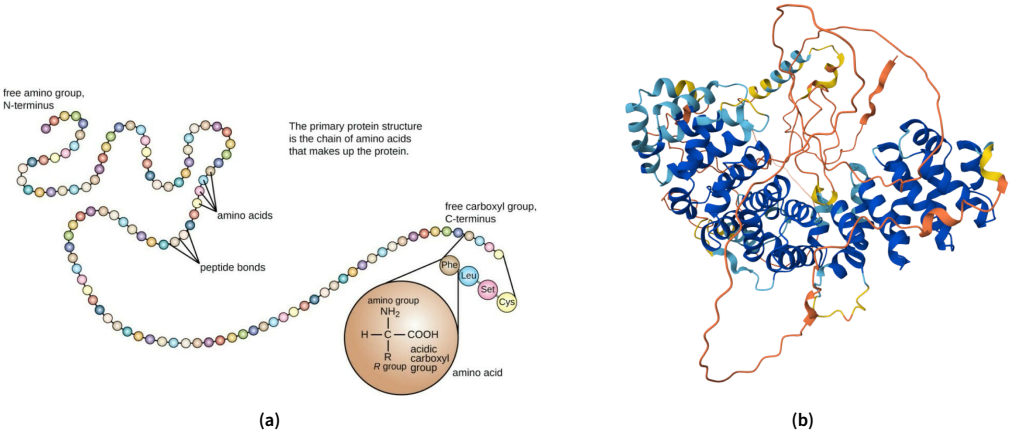


Figure 1. **(a)** An amino acid sequence with residue R shown in the leading amino acid. Image by OpenStax Microbiology, Wikimedia Commons, licensed under CC BY 4.0. **(b)** A tertiary structure predicted by AlphaFold for PKP1. Image by BQUB24-Ebahr, Wikimedia Commons, licensed under CC BY-SA 4.0.

poses a great challenge to this characterization. The primary initiative of protein function prediction is then to develop tools to generate the GO terms associated to a protein from more rudimentary data such as the primary, secondary, tertiary, or quaternary structure. The transformer architecture proposed in (Gu et al. 2023) and reimplemented here, generates these predictions from data reliant on all four levels of structures.

1.2 Scope & Scale

Both in (Gu et al. 2023) and in the present work, the model is trained, validated, and tested on a dataset containing 2748 GO terms—the label set. While the original implementation trained on more than 220k protein structures, the reimplementation here was trained, validated, and tested on 28,289, 3,135, and 3,123 structures, respectively. Before input into the model, each protein is mapped to an (undirected) attributed graph, i.e., a graph with a vector in $\mathbb{R}^F$ for each node. This representation, referred to as the protein graph representation, has nodes corresponding to residues in the primary structure, edges between residues with a distance < 10 Å between their C_α atoms (in the 3D structure), and vectors concatenated from the one-hot amino acid label (20-dimensional) and the ESM1b[1] embedding (1280-dimensional).

The present dataset, after computing the relevant protein graphs, contains an average of 261 nodes and 4566 edges per protein. From this, we estimate a total of 9M nodes across the dataset, each with a corresponding 1300-dimensional vector, and a total of 158M edges.

2. The HEAL Architecture

The HEAL model takes as input, the aforementioned 1300-dimensional protein graph representation and runs it through (1) A Graph Convolution Network (GCN Encoder); (2) a Hierarchical Graph Transformer (HGT); and ultimately (3) a Multilayer Perceptron (MLP). Distinct from the model itself, but incorporated into this architecture is (1) the preprocessing with EMS1b and the RCSB Protein Databank (PDB); (2) a contrastive learning scheme; and (3) binary cross-entropy with an added contrastive term as the loss function. Below, we briefly describe each of these elements, although we save further discussion of the preprocessing for Section 3.

1. Meta's protein language model (Rives et al. 2021)

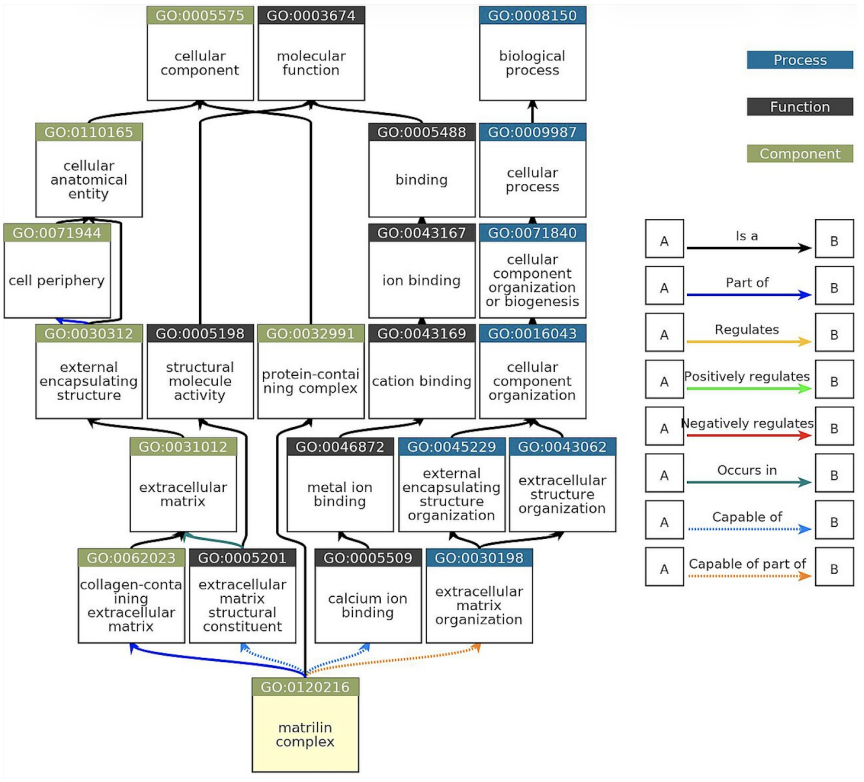


Figure 2. The top level of the Gene Ontology as it descends to those terms related to family of matrilin complex protein structures. Image by EMBL's European Bioinformatics Institute, Wikimedia Commons, released under the CC0 1.0 Public Domain Dedication.

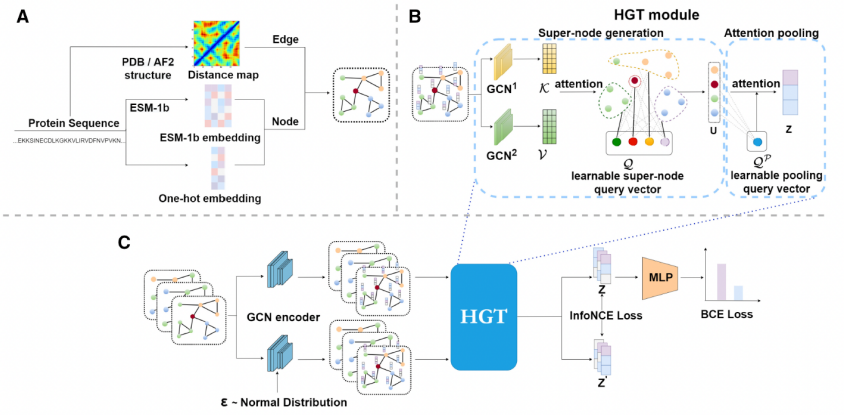


Figure 3. Overview of the HEAL architecture. **A** Attributed graph representation mining. **B** The Hierarchical Graph Transformer. **C** The HEAL model with contrastive learning. Image from Gu et al. (Gu et al. 2023), originally published in *Bioinformatics* and distributed under the https://creativecommons.org/licenses/by/4.0/Creative Commons Attribution 4.0 License.

2.1 The Compositional Model

In what follows, we represent an n-node, m-dimensional protein graph representation as an adjacency matrix $A \in \mathbb{R}^{n \times n}$ together with matrix $H \in \mathbb{R}^{n \times m}$. Here, the i^{th} row of H represents the vector for

the i^{th} node.

2.1.1 Graph Convolution Network

The first component of the HEAL model, the GCN encoder, acts on graph representations computed via the ESM-1b and PDB in the preprocessing stage, denoted by $H^{(0)} \in \mathbb{R}^{L\times F}$ where, in practice, the feature dimension is $F = 1300$. The graph representation is then updated inductively as follows:

$$H^{(n+1)} = \text{ReLU}\left(\tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}H^{(n)}W^{(n)}\right) \tag{1}$$

where $\tilde{A} = A + I$ and $\tilde{D}^{-0.5}$ are the self-loop adjacency and the normalized degree matricies, respectively, i.e., $\tilde{D}_{ii}^{-0.5} = \deg(i)^{-0.5}$ and 0 otherwise. After some fixed number of updates, the resulting $H^{(N)}$, denoted $\text{GCN}(H^{(0)}, A)$, is passed to the transformer block. For motivation and interpretation of this scheme, see (Kipf and Welling 2017).

2.1.2 Hierarchical Graph Transformer

Given the output H of the GCN encoder, we construct the key, and value representations ($\mathcal{K}$, and $\mathcal{V}$, respectively) by introducing two additional GCN modules:

$$\mathcal{K} = \text{GCN}_1(H, A) \in \mathbb{R}^{L\times D}, \quad \mathcal{V} = \text{GCN}_2(H, A) \in \mathbb{R}^{L\times D} \tag{2}$$

In contrast to this, the query representation is defined as a single trainable matrix $\mathcal{Q} \in \mathbb{R}^{K\times D}$ where K is referred to as the number of super-nodes. In the original implementation, K was taken to be ?. From here, the super-node embedding matrix is computed as

$$U = \text{MLP}(\Gamma); \quad \Gamma := \text{softmax}\left(\frac{\mathcal{Q}\mathcal{K}^\top}{\sqrt{D}}\right)\mathcal{V} \tag{3}$$

where MLP is a multilayer perceptron. The utility of the MLP increases dramatically in the multi-head framework presented in (Gu et al. 2023), however, we omit a discussion of this generalization as it was not implemented in the present work.

For the final step of the transformer block, the authors propose the attention pooling scheme seen below.

$$z = \text{softmax}\left(\frac{\mathcal{Q}^P(U\mathcal{K}^P)^\top}{\sqrt{D}}\right)U\mathcal{V}^P \tag{4}$$

Here, $\mathcal{K}^P$, $\mathcal{V}^P \in \mathbb{R}^{D\times D}$, and $\mathcal{Q}^P \in \mathbb{R}^{1\times D}$ carry additional training parameters.

2.1.3 Multilayer Perceptron

To interpret the HGT output, the authors conclude with a sigmoid-activated MLP obtaining a GO term prediction

$$\hat{y} = \text{MLP}(z) \in \mathbb{R}^C \tag{5}$$

whose entries, which take values in $(0, 1)$, are interpreted as the probability that the given protein is related to the relevant GO term.

2.2 Contrastive Learning & Losses

To aid in the learning process, the architecture incorporates contrastive learning as seen in (He et al. 2019). Below, we present a slightly modernized version in which the initial feature matrix, $H^{(0)}$, is perturbed twice in arbitrary directions and the InfoNCE loss function is computed on the HGT outputs of the perturbed pair (Chen et al. 2020) (You et al. 2020). More precisely, given our initial attributed graph $(H^{(0)}, A)$, we perturb each feature vector $h_i = (H^{(0)}_{i1}, \ldots, H^{(0)}_{iF})$ by random vectors v_i^1 and v_i^2, both with ℓ^2 norm equal to ε, obtaining

$$h_i^1 := h_i + v_i^1\,, \quad h_i^2 = h_i + v_i^2 \tag{6}$$

and the corresponding perturbed feature matrices $H_1^{(0)}$ and $H_2^{(0)}$. We then pass the triple $(H^{(0)}, H_1^{(0)}, H_2^{(0)})$ to the model and store the HGT output (z, z_1, z_2) (Equation 4) to later compute the contrastive loss.

Before presenting the objective functions, let us assume that the training is being conducted on batches and the size of the present batch is M. We then index all variables by $m = 1, \ldots M$ to refer to that variable as it pertains to the m^{th} graph in the batch. Furthermore, for convenience, we define $Z, Z_1, Z_2 \in \mathbb{R}^{M\times D}$ to be the collection of the z, z_1, and z_2 representations (organized into a vertical concatenation) while $\hat{y}_m$ and y_m denote the predicted and actual GO labels for the m^{th} graph, respectively. Now, the total loss implemented here is the standard supervised binary cross-entropy with an additional contrastive term:

$$\mathcal{L}_{\text{tot}} := \mathcal{L}_{\text{sup}} + \lambda\mathcal{L}_{\text{con}}\,; \tag{7}$$

In particular, we have

$$\begin{aligned}\mathcal{L}_{\text{sup}} &:= -\frac{1}{M\cdot C}\sum_{l=1}^{C}\sum_{m=1}^{M}\Big(y_{ml}\log(\hat{y}_{ml}) + (1-y_{ml})\log(1-\hat{y}_{ml})\Big)\\ \mathcal{L}_{\text{con}} &:= \frac{1}{2}\big(\mathcal{L}_{\text{NCE}}(Z_1, Z_2) + \mathcal{L}_{\text{NCE}}(Z_2, Z_1)\big)\end{aligned} \tag{8}$$

where $\mathcal{L}_{\text{NCE}}$ is the standard InfoNCE loss function, i.e.,

$$\mathcal{L}_{\text{NCE}}(Z, Z') = -\frac{1}{M}\sum_{m=1}^{M}\log\frac{\exp\big(z_m \star z'_m/\tau\big)}{\sum_{m'=1}^{M}\exp\big(z_m \star z'_{m'}/\tau\big)}\,, \tag{9}$$

with $\star$ denoting cosine similarity. We remark that the contrastive loss cannot cannot be decomposed into a summation of graph-level terms and thus, is inherently a batch-level entity.

3. The Julia Implementation

In what follows, we first discuss the machine learning stack and the type theoretic approach that is inherent to Julia. We then present our reimplementation, HEALJ, in terms of its hyperparameters and report a preliminary experiment that informed our contrastive hyperparameter decisions. We conclude this section with a final analysis of the test results upon completion of the full training.

Aside from learned parameters and preprocessed attributed graph data, the full repository can be found at ((M.M. 2025)). The file layout for this repository, depicted in Figure 4, is organized as follows:

1. all sequence and PDB data, ESM-1b weights (gitignored), Python preprocessing scripts and preprocessed attributed graph data are under `/preprocessing/`;

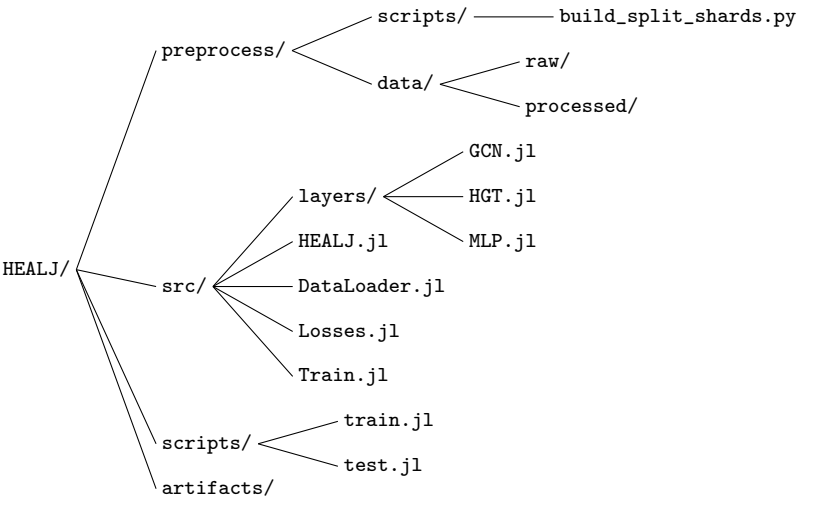


Figure 4. HEALJ Repository file tree (M.M. 2025).

2. all type constructions and critical functions, including metadata for the attributed graphs and model definitions, are stored under `/src/`;
3. training and testing, which relies extensively on `/src/`, is executed from `/scripts/`;
4. HEALJ weights (gitignored) are pushed to `/artifacts/`; and
5. data analysis is carried out under `/analysis/`.

3.1 Preprocessing & Metadata

The protein dataset used here is a subset of that used in the original implementation and was first aggregated and decomposed into a train/validate/test split by the authors of (Gligorijević et al. 2021). This dataset consists of FASTA files storing, for each protein, the sequence data along with the relevant GO terms out of a fixed label-set (containing a total of 2748 terms). The 3D structure data was downloaded as needed throughout the preprocessing via `rcsb.org`[2] and the initial ESM-1b weights were downloaded via `fbaipublicfiles.com` as seen in the repository's `README`. With these capabilities, the attributed protein graphs were mined in 6 hours[3] and stored, with their GO labels, as distinct groups in HDF5 files under `/preprocess/data/processed/shards` via the script `build_split_shards.py`.

Each group in the HDF5 files, i.e., each protein in the dataset, carries the following attributes:

1. `native_x`, a node-feature matrix in which each row is a one-hot encoding of the amino acid identity at the corresponding residue;
2. `x`, a node-feature matrix whose rows are ESM-1b embeddings of the corresponding residues;
3. `y`, a multi-hot label vector over the fixed GO label-set; and
4. `edge_index`, encoding the adjacency matrix as a $2 \times |E|$ matrix whose rows represent the source and destination mappings from edges to node.

We remark that the edge index vectors are used in place of a sparse adjacency matrix for optimal storage and to simplify the disjoint union operation (in the category of graphs) that is necessarily implemented when batching. Explicitly, the $(1, i)$ and $(2, i)$ entries of `edge_index` are integers

2. see line 87 of `/preprocess/scripts/build_split_shards.py`
3. on an NVIDIA GeForce RTX 5060

corresponding to the indices of the rows in `native_x` (or equivalently, `x`) that represent the source and target of the i^{th} edge, respectively.

To reason about the dataset stored across many HDF5 files (containing 34,547 attributed protein graphs totaling 41 Gigabytes) and build batches, we constructed four core Julia types defined in the DataLoader module (`src/DataLoader.jl`):

1. GraphRef, whose objects point to a single group (i.e., an attributed protein graph) inside a single shard while storing the number of nodes in the referenced graph;
2. Dataset, whose objects carry a Vector{GraphRef} object[4] used as an index for the dataset, along with the label-set size and feature dimension used in the throughout the dataset;
3. Datasets, whose objects store three Dataset objects: train, validate, and test; and
4. MicroBatch, whose objects represent a batch of graphs graph_refs :: Vector{GraphRef} together with
   (a) the batched node-feature matrix $H^{(0)}$ concatenated (vertically) from that of all batched graphs,
   (b) edge index vectors src and dst for the aggregate graph obtained via a disjoint union of the batched graphs,
   (c) a label matrix γ whose rows represent the label vectors for the given node, and
   (d) an assignment vector node2graph whose i^{th} entry stores the graph_ref index corresponding to the graph containing the i^{th} node in the aggregate graph.

3.2 The Model

HEALJ was constructed and trained using the Julia-native machine learning framework defined by Flux.jl, Zygote.jl, CUDA.jl, and Optimizers.jl. To this project, these libraries provide (1) model construction with trainable/not-trainable metadata, (2) automatic differentiation, (3) device handling and GPU optimization, and (4) the Adam Optimizer, respectfully. In what follows, we present the details of this construction and training.

3.2.1 Structural Hyperparameters

This model was written extensibility and can be trained with any suitable combination of hyperparameters. A preliminary experiment was conducted to find the optimal contrastive hyperparameters $(\tau, \lambda, \varepsilon)$ defined in Section 2.2. However, this experiment along with the main training were all conducted with the structural ensemble described below.

Recall the variables presented in Section 2.1. We begin with $F = 1300$ obtained from the concatenation of the one-hot amino acid label and the ESM-1b embedding. For the hidden dimension D, we chose 256 and used $K = 16$ super nodes. The GCN Encoder consisted of two layers $\mathbb{R}^F \to \mathbb{R}^D \to \mathbb{R}^D$ while the key and value GCNs, GCN_1 and GCN_2, each consisted of a single layer $\mathbb{R}^D \to \mathbb{R}^D$. All GCN layers omitted bias. The HGT block utilized the single-head scheme and the internal MLP was a single layer $\mathbb{R}^D \to \mathbb{R}^D$. The final MLP was composed of two layers $\mathbb{R}^D \to \mathbb{R}^{512} \to \mathbb{R}^{C=2748}$.

3.2.2 Contrastive Hyperparameters

To optimize the contrastive hyperparameters $h = (\tau, \lambda, \varepsilon)$ seen in Section 2.2, preliminary training was conducted on a data subset with a 1000/328/347 train/validate/test split over 10 epochs. This training was executed for all 27 combination generated by

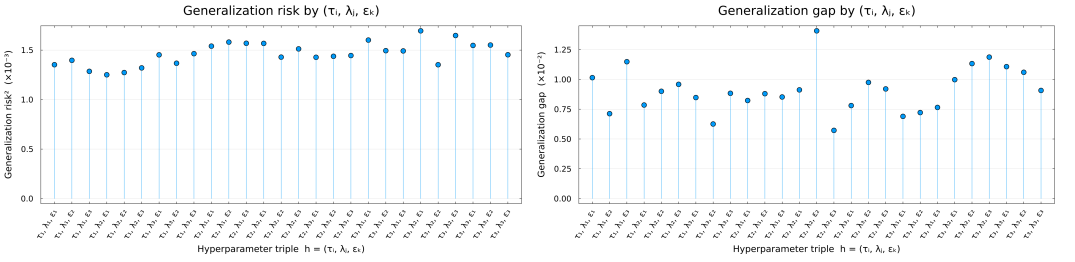
$$\tau \in \{0.1, 0.5, 1.0\}, \quad \lambda \in \{0.01, 0.1, 1.0\}, \quad \text{and} \quad \varepsilon \in \{0.001, 0.01, 1.0\}. \tag{10}$$

4. i.e., a vector of GraphRef objects

The total training time (3:39 hours) was logged for estimating the full training. For each configuration h, we selected the *best epoch* $e^\star(h) = \arg\min_e \mathcal{L}^{\text{sup}}_{\text{val}}(e; h)$. We then reported

$$\text{Risk}(h) = \mathcal{L}^{\text{val}}_{\text{sup}}(e^\star(h); h), \qquad \text{Gap}(h) = \mathcal{L}^{\text{val}}_{\text{sup}}(e^\star(h); h) - \mathcal{L}^{\text{sup}}_{\text{train}}(e^\star(h); h),$$

aligned with (?)[5]. The optimal configuration was then chosen qualitatively with Risk as the primary metric and Gap as used to differentiate between the five configurations with the lowest Risk. The results shown in Figure 5 suggest $h = (0.1, 0.1, 0.001)$ as the optimal contrastive hyperparameters and thus, this is what was used in the full training.



(a) **(b)**

As a final remark, we note that while $\varepsilon = 0.001$ appears quite low, all 1300 vectors in the attributed protein graph are perturbed by a vector of this magnitude.

Figure 5. Primary metrics associated to each contrastive hyperparameter configuration in the preliminary experiment. The resulting contrastive hyperparameter decision was $h = (\tau_1, \lambda_2, \varepsilon_1) = (0.1, 0.1, 0.001)$ (M.M. 2025).

3.3 Results & Analysis

The full training on the 28,289 attributed protein graphs was initiated with the structural and contrastive hyperparameters fixed as described above. The process was completed in 12 hours on an NVIDIA GeForce RTX 5060. The training plan was set to run for 25 epochs as it was assumed that no overfitting would occur prematurely. The total loss that was computed per epoch suggests that this assumption was correct (see Appendix Appendix 1).

The learned parameters were then tested on the aforementioned 3,123 protein test set. To summarize the performance in a manner conducive to comparison with the original implementation, we report the $F_{\max}$ metric expounded in (Jiang, Oron, Clark, et al. 2016) and constructed below. First, note that the model output $\hat{y}$ has entries with values in $(0, 1)$. To obtain boolean predictions, we introduce the threshold $t \in [0, 1]$ defining the map $\hat{y} \mapsto \hat{y}_{\text{bool}}$. Then we obtain a predicted term set $P_i(\tau)$ for each protein i, and compute

$$\text{pr}(t) = \frac{1}{m(t)} \sum_{i=1}^{m(t)} \frac{|P_i(t) \cap T_i|}{|P_i(t)|}, \qquad \text{rc}(t) = \frac{1}{n_e} \sum_{i=1}^{n_e} \frac{|P_i(t) \cap T_i|}{|T_i|}, \tag{11}$$

where T_i is the true term set, $m(t)$ is the number of proteins with at least one predicted term at threshold t, and n_e is the number of evaluated proteins. The desired metric is then

$$F_{\max} = \max_{\tau} \left\{ \frac{2\,\text{pr}(t)\,\text{rc}(t)}{\text{pr}(t) + \text{rc}(t)} \right\}. \tag{12}$$

5. https://arxiv.org/html/2312.13842v1

In our testing, HEALJ achieved an score of $F_{\max} = 0.4574$ (attained at $t^* = 0.04$). While we cannot directly compare this metric to that in the original implementation as the authors computed this metric for the GO terms under molecular_function, biological_process, and cellular_component separately, our values appear to be aligned. In particular, the original implementation obtained $(F^{\mathrm{MF}}_{\max}, F^{\mathrm{BP}}_{\max}, F^{\mathrm{CC}}_{\max}) = (0.747, 0.595, 0.687)$, and our value sits comfortably in (Gu et al. 2023, Table 1) presenting these metrics as they pertain to other leading models.

4. Conclusion

The present work introduced HEALJ, a Julia reimplementation of the HEAL architecture for protein function prediction, together with the full preprocessing and data-loading framework built around ESM-1b embeddings and PDB-derived structures. We defined the compositional model, including the GCN encoder, hierarchical graph transformer, and MLP head in the Julia/Flux ecosystem, and incorporated a symmetric InfoNCE contrastive loss. On a subset of 28,289 training proteins (with 3,135 validation and 3,123 test proteins), the resulting model trained stably for 25 epochs on a single NVIDIA GeForce RTX 5060 and achieved an overall $F_{\max} = 0.457$ at threshold $t = 0.04$. This is aligned with those values seen in the literature, but significantly outperformed by the original HEAL implementation.

Several limitations and generalizations are left for future work. First, the present study operated on a smaller dataset than the original and utilized only the single-head attention scheme; incorporating the full dataset and implementing multi-head attention are trivial next steps. Further, our contrastive hyperparameter experiment was not exhaustive and our interpretable metrics (Appendix Appendix 2) were only evaluated at a fixed threshold $t = 0.5$.

This Julia implementation is representative of the growing movement towards lightweight stacks with advanced capabilities and type-theoretic foundations. In this sense, HEALJ serves as both a reproducing reference and an extensible platform for future innovation.

References

Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th international conference on machine learning (icml).* arXiv: 2002.05709 [cs.LG]. https://arxiv.org/abs/2002.05709.

Gene Ontology Consortium. 2025. Gene ontology resource: release statistics. Current release 2025-10-10: 39,354 GO terms, 9,281,704 annotations, 1,601,555 gene products, 5,495 species, October 10, 2025. https://geneontology.org/.

Gligorijević, Vladimir, P. Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Daniel Berenberg, Tommi Vatanen, Chris Chandler, et al. 2021. Structure-based protein function prediction using graph convolutional networks. *Nature Communications* 12 (1): 3168. ISSN: 2041-1723. https://doi.org/10.1038/s41467-021-23303-9. https://doi.org/10.1038/s41467-021-23303-9.

Gu, Zhonghui, Xiao Luo, Jiaxiao Chen, Minghua Deng, and Luhua Lai. 2023. Hierarchical graph transformer with contrastive learning for protein function prediction. *Bioinformatics* 39 (7): btad410. https://doi.org/10.1093/bioinformatics/btad410. https://academic.oup.com/bioinformatics/article/39/7/btad410/7208864.

He, Kaiming, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. 2019. Momentum contrast for unsupervised visual representation learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* 9726–9735. https://api.semanticscholar.org/CorpusID:207930212.

Jiang, Yuxiang, Tal Ronnen Oron, Wyatt T. Clark, et al. 2016. An expanded evaluation of protein function prediction methods shows an improvement in accuracy. *Genome Biology* 17 (184). https://doi.org/10.1186/s13059-016-1037-6.

Kipf, Thomas N., and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International conference on learning representations.* https://openreview.net/forum?id=SJU4ayYgl.

M.M. 2025. *Healj.* https://github.com/monte-mahlum/HEALJ. Accessed 2025-12-13.

Rives, Alexander, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, et al. 2021. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences* 118 (15): e2016239118. https://doi.org/10.1073/pnas.2016239118. https://www.pnas.org/doi/10.1073/pnas.2016239118.

You, Yuning, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. In *Advances in neural information processing systems,* edited by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, 33:5812–5823. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/3fe230348e9a12c13120749e3f9fa4cd-Paper.pdf.

Appendix 1. Training and Validation Loss

Table 1. Training and validation loss history for the full HEALJ run.

Epoch	Train total	Train sup	Train con	Val sup
1	0.045360	0.038035	0.073248	0.024304
2	0.023752	0.022055	0.016961	0.021691
3	0.021221	0.019824	0.013967	0.020579
4	0.019707	0.018424	0.012837	0.019696
5	0.018742	0.017600	0.011419	0.019268
6	0.017741	0.016683	0.009807	0.018989
7	0.016967	0.015983	0.008918	0.018907
8	0.016292	0.015366	0.008372	0.018883
9	0.015712	0.014866	0.007760	0.018714
10	0.015215	0.014420	0.007363	0.018628
11	0.014800	0.014030	0.006974	0.018604
12	0.014392	0.013644	0.006748	0.018574
13	0.013998	0.013272	0.006404	0.018606
14	0.013654	0.012942	0.006241	0.018612
15	0.013335	0.012640	0.006066	0.018624
16	0.012999	0.012323	0.005769	0.018604
17	0.012664	0.011998	0.005654	0.018601
18	0.012370	0.011714	0.005515	0.018578
19	0.012112	0.011474	0.005494	0.018570
20	0.011784	0.011159	0.005365	0.018579
21	0.011522	0.010914	0.005311	0.018583
22	0.011313	0.010728	0.005245	0.018618
23	0.010982	0.010421	0.005145	0.019120
24	0.010783	0.010253	0.004965	0.019709
25	0.010477	0.009973	0.004906	0.019667

Appendix 2. Interpretable Metrics

For further interpretability, we here present five additional metrics each more approachable than $F_{\max}$. To begin, we fix the threshold $t = 0.5$ and compute the true per-protein GO term count:

$$n_{\gamma 1} \sum_{i=1}^{C} \mathbf{1}\{\gamma_i = 1\}, \quad n_{\gamma 0} = \sum_{i=1}^{C} \mathbf{1}\{\gamma_i = 0\}. \tag{13}$$

together with the positive-positive, false positive, false negative, and negative-negative counts:

$$\begin{aligned} n_{\text{pp}} &= \sum_{i=1}^{C} \mathbf{1}\{\hat{\gamma}_i = 1, \gamma_i = 1\}, \quad n_{\text{pn}} = \sum_{i=1}^{C} \mathbf{1}\{\hat{\gamma}_i = 1, \gamma_i = 0\}, \\ n_{\text{np}} &= \sum_{i=1}^{C} \mathbf{1}\{\hat{\gamma}_i = 0, \gamma_i = 1\}, \quad n_{\text{nn}} = \sum_{i=1}^{C} \mathbf{1}\{\hat{\gamma}_i = 0, \gamma_i = 0\}. \end{aligned} \tag{14}$$

Using these quantities, we defined five per-protein metrics

$$\begin{aligned} m_1 &= \frac{n_{\text{pp}}}{n_{y1}}, & m_2 &= \frac{n_{\text{pn}}}{n_{y1}}, \\ m_3 &= \frac{n_{\text{np}}}{n_{y0}}, & m_4 &= \frac{n_{\text{nn}}}{n_{y0}}, \\ m_5 &= \left| n_{y1} - n_{\hat{y}1} \right|. \end{aligned} \tag{15}$$

which can be interpreted as follows:

1. (m_1) the ratio of correctly predicted GO terms to the true number of GO terms
2. (m_2) the ratio of false positives to the true number of negatives,
3. (m_3) the ratio of false negatives to the true number of positives,
4. (m_4) the ratio of correctly predicted unrelated GO terms to the true number of unrelated GO terms, and
5. (m_5) the positive difference between the number of true and predicted GO terms.

Finally, for each metric m_r we report the mean and sample variance across proteins:

$$\bar{m}_r = \frac{1}{N} \sum_{p=1}^{N} m_r^{(p)}, \qquad s_r^2 = \frac{1}{N-1} \sum_{p=1}^{N} \left(m_r^{(p)} - \bar{m}_r \right)^2, \tag{16}$$

where $N = 3,123$ is the total number of proteins in the test set.

The results, reported in Table 2, must be read with the context that the average number of GO terms per protein in the test set, n_{y1}, is extremely low relative to the number of unrelated terms (n_{y0}). In fact, the average number of GO terms across the test set, $\bar{n}_{y1}$, was computed to be 88.6. With this context, the metrics seem to imply that the learned parameters are poorly predicting protein function. However, our $F_{\max}$ analysis suggests that $t = 0.04$ is a significantly better choice. Future work involves running this same analysis with $t = 0.04$.

Table 2. Protein-level summary statistics on the test set. Means and sample variances are computed across proteins; metrics with denominators are computed only when the denominator is nonzero.

Metric	Mean	Sample var	StD
$m_1 = \frac{n_{\text{pp}}}{n_{y1}}$	0.216	0.055	0.235
$m_2 = \frac{n_{\text{pn}}}{n_{y1}}$	0.091	0.102	0.319
$m_3 = \frac{n_{\text{np}}}{n_{y0}}$	0.029	0.001	0.032
$m_4 = \frac{n_{\text{nn}}}{n_{y0}}$	0.999	0.000	0.003
$m_5 = \lvert n_{y1} - n_{\hat{y}1} \rvert$	72.489	7102.477	82.276