BIOBLOBS: Differentiable Graph Partitioning for Protein Representation Learning

Xin (Allen) Wang
Yale University
New Haven, CT
allen.wang.xw532@yale.edu

Carlos Oliver
Vanderbilt University
Nashville, TN
carlos.oliver@vanderbilt.edu

Abstract

Protein function is driven by coherent substructures that vary in size and topology, yet current protein representation learning models (PRL) distort these signals by relying on rigid substructures such as k-hop and fixed radius neighborhoods. We introduce BIOBLOBS, a plug-and-play, fully differentiable module that represents proteins by dynamically partitioning structures into flexibly sized, non-overlapping substructures ("blobs"). The resulting blobs are quantized into a shared and interpretable codebook, yielding a discrete vocabulary of function-relevant protein substructures used to compute protein embeddings. We show that BIOBLOBS's representations improve the performance of widely used protein encoders such as GVP-GNN across various PRL tasks. Our approach highlights the value of architectures that directly capture function-relevant protein substructures, enabling both improved predictive performance and mechanistic insight into protein function.

Source code: https://github.com/OliverLaboratory/BioBlobs

1 Introduction

Existing protein representation learning (PRL) systems often operate over rigidly defined "atoms" [9, 10], yet protein function is driven by cohesive substructures that operate modularly [23]. Among many examples are catalytic triads (Ser–His–Asp), Rossmann-like nucleotide-binding cores, and P-loop (Walker A/B) NTPase sites [8, 24, 18]. When the unit is fixed *a priori*—e.g., k-hop neighborhoods or voxel grids—models can fracture functional assemblies and distort the learned representations. This would be like trying to understand the mechanisms of a bicycle by cutting it into uniformly sized cubes. Likewise, in proteins, functional modules rarely adopt such regular substructures. Compounding this, the appropriate units are context-dependent and vary in size and topology. Moreover, the atomization step is inherently discrete (e.g., selecting a residue and its neighbors), making it non-differentiable and therefore typically excluded from end-to-end training. The atoms we seek should be (1) connected substructures of the protein, (2) contributory to function, and (3) distributed modularly, reflecting the conservative forces of evolution.

We cast the problem of atomizing a protein into cohesive substructures as a graph-partitioning task [6]. Protein graphs—capturing neighborhood relations and spatial organization among residues or atoms—are standard in PRL, with GVP-GNNs being a widely used example [15]. Graph partitioning assigns each node to exactly one part, producing a disjoint cover of the graph; on residue/atom graphs, such partitions yield non-overlapping groups—precisely the structural units we aim to model. Because the space of partitions grows exponentially with the number of nodes, many natural formulations are NP-hard [11]. Nonetheless, recent advances in neural combinatorial optimisation over graphs learn probability distributions on nodes/edges to construct powerful probabilistic algorithms for tasks such as set cover, alignment, and compression [16, 5]. Here we introduce BIOBLOBS, which

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

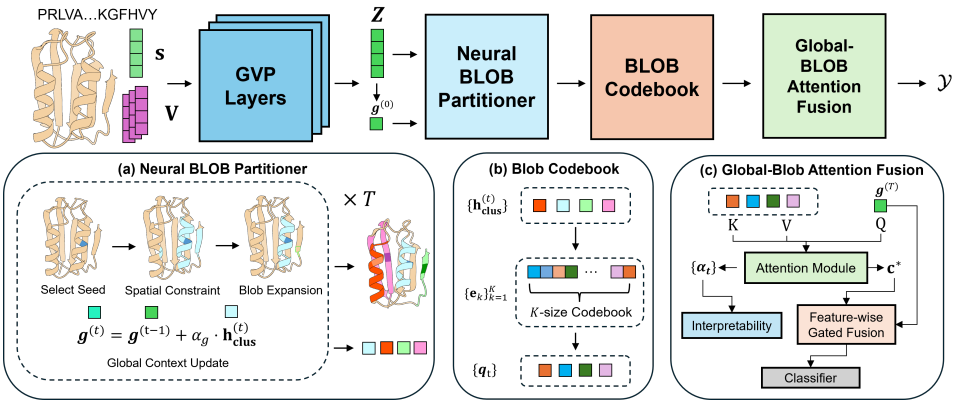

Figure 1: Overview of the BIOBLOBS pipeline. A GVP encoder generates residue embeddings, which are processed by three modules. **(a) Neural Blob Partitioner:** Iteratively selects a seed residue, identifies a candidate pool from its k-hop neighborhood, and expands to form a cohesive substructure (a blob). **(b) Blob Codebook:** Quantizes blob embeddings into a discrete vocabulary of common, function-relevant substructures. **(c) Global–Blob Attention Fusion:** Integrates quantized blob representations with a global protein feature using attention, yielding a final embedding and interpretable importance scores for each blob.

performs principled partitioning of 3D protein graphs into cohesive substructures to build strong protein embeddings. BIOBLOBS *boosts GVP-GNNs to achieve top performance on three established protein-function benchmarks under rigorous controls for structure leakage, and unlocks a new way of systematically studying function-critical protein substructures.*

2 BIOBLOBS

The BIOBLOBS layer dynamically computes a hard assignment of residues to distinct substructures (sets of residues), which we term "blobs". We then compute representations of each blob and incorporate global context to arrive at a final embedding for the whole protein.

2.1 Protein Graph Construction

Each protein is represented as a geometric graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where nodes $v_i \in \mathcal{V}$ are amino acid residues. Edges $(i, j) \in \mathcal{E}$ connect spatially proximate residues, identified by the k-nearest neighbors of their C_α atoms. Each node v_i has geometric features $\mathbf{h}_i = (\mathbf{s}_i, \mathbf{V}_i)$, derived from its O, C, C_α, N atom coordinates. These include scalar features $\mathbf{s}_i \in \mathbb{R}^{d_s}$ (e.g., dihedral angles) and vector features $\mathbf{V}_i \in \mathbb{R}^{d_v \times 3}$ (e.g., orientations). Edges have features $\mathbf{h}_{ij}^E = (\mathbf{s}_{ij}, \mathbf{V}_{ij})$, which include scalars $\mathbf{s}_{ij} \in \mathbb{R}^{d_e}$ (e.g., distances) and vectors $\mathbf{V}_{ij} \in \mathbb{R}^{d_{ev} \times 3}$ (e.g., directions).

2.2 Protein Structure Encoder

We use Geometric Vector Perceptron (GVP) layers [15] to compute geometrically equivariant residue features. The structure encoder consists of L GVP convolution layers:

$$\mathbf{h}_i^{(\ell+1)} = \text{GVPConv}\left(\mathbf{h}_i^{(\ell)}, \left\{\mathbf{h}_j^{(\ell)} : j \in \mathcal{N}(i)\right\}, \mathbf{h}_{ij}^E\right) \tag{1}$$

where $\mathcal{N}(i)$ are the neighbors of node i. After the final layer, a projection layer produces scalar node features $\mathbf{z}_i = \text{Proj}\left(\mathbf{s}_i^{(L)}\right) \in \mathbb{R}^D$. These features are then passed to the partition module.

2.3 Neural BLOB Partitioner

Given residue embeddings $\mathbf{Z} \in \mathbb{R}^{N \times D}$, an initial global context vector $\mathbf{g}^{(0)}$ is computed by pooling all residue features. The partitioner then iteratively constructs up to T blobs. An assignment matrix $\mathbf{M} \in \{0, 1\}^{N \times T}$ records the residue composition of each blob, and an availability mask $\mathbf{m} \in \{0, 1\}^N$ tracks unassigned residues to ensure the partitions are non-overlapping. Each iteration performs a discrete, differentiable partition [5] using three key steps. Full algorithm details are shown in Alg. 1.

Seed Residue Selection. We begin by using a seed-scoring network to predict logits over all residues: $\boldsymbol{\ell}^{(t)} = f_{\text{seed}}([\mathbf{Z}; \mathbf{g}^{(t-1)}])$. To ensure only unassigned residues are considered, we mask unavailable candidates by setting their logits to $-\infty$ using the mask $\mathbf{m}^{(t-1)}$. We then sample a seed using the Gumbel-Softmax [14], which adds Gumbel noise to the masked logits and applies a softmax function with a gradually decreasing temperature $\tau^{(t)}$. This annealing process shifts sampling from exploratory to deterministic during training. To obtain a discrete seed while maintaining differentiability, we use the straight-through (ST) estimator [3]. In the forward pass, we select the seed using a discrete one-hot vector, $\mathbf{e}^{(t)}$, derived from the $\arg\max$ of the Gumbel-Softmax probabilities $\mathbf{w}^{(t)}$. For the backward pass, however, we use the soft probabilities $\tilde{\mathbf{w}}^{(t)}$ directly to allow gradients to flow. This creates a final seed embedding, $\mathbf{z}_{\text{seed}}^{\text{ST}}$, which acts as a continuous proxy for the discrete selection and enables end-to-end training.

Threshold Prediction. Once the seed residue $v_{\text{seed}}^{(t)}$ is chosen, we define a candidate set for expansion, $\mathcal{R}^{(t)}$, by selecting the available residues within its k-hop neighborhood. A learned threshold head then predicts a blob-specific threshold, $\theta^{(t)}$, using the seed's embedding ($\mathbf{z}_{\text{seed}}^{\text{ST}}$), the global context ($\mathbf{g}^{(t-1)}$), and local structural statistics ($\boldsymbol{\phi}^{(t)}$) from the candidate set. This adaptive threshold determines how many candidates are added to the blob, which regulates its final size during the expansion step.

Blob Expansion. For each candidate residue in the set $\mathcal{R}^{(t)}$, we first compute a seed-conditioned attention score, $s_i^{(t)}$, using a scaled dot-product. These scores are converted into inclusion probabilities $p_i^{(t)}$, which are thresholded to get discrete binary memberships $c_i^{(t)}$. To ensure the process remains differentiable, we use a straight-through (ST) correction $\tilde{p}_i^{(t)}$:

$$p_i^{(t)} = \sigma\big((s_i^{(t)} - \theta^{(t)})/\tau^{(t)}\big), \quad c_i^{(t)} = \mathbb{I}[p_i^{(t)} \geq 0.5], \quad \tilde{p}_i^{(t)} = c_i^{(t)} + (p_i^{(t)} - \text{sg}(p_i^{(t)})) \tag{2}$$

To prevent blobs from becoming too large, we limit their size to a maximum of S candidates, selecting those with the highest inclusion probabilities. The final blob embedding, $\mathbf{h}_{\text{clus}}^{(t)}$, is then formed using an ST pooling method that combines a discrete mean (for the forward pass) with a soft mean (for backpropagation). Lastly, the global context is updated with the blob's information via a residual connection, $\mathbf{g}^{(t+1)} = \mathbf{g}^{(t)} + \alpha_g \, f_{\text{global}}(\mathbf{h}_{\text{clus}}^{(t)})$, and the assignment matrix is updated.

2.4 Substructure Codebook

After the partitioner, we obtain blob embeddings $\mathbf{H} = [\mathbf{h}_{\text{clus}}^{(1)}, \ldots, \mathbf{h}_{\text{clus}}^{(T)}] \in \mathbb{R}^{T \times D}$. We introduce a vector–quantization codebook $\mathbf{E} = \{\mathbf{e}_k\}_{k=1}^{K} \subset \mathbb{R}^D$ to map each blob to a discrete substructure token [26, 22], steering the encoder and partitioner to discover function–relevant substructures shared across the dataset. Let $\mathbf{h}_t$ denote the t-th blob embedding, the nearest–neighbor quantization selects its token as

$$k_t \;=\; \arg\min_{k \in \{1,\ldots,K\}} \|\mathbf{h}_t - \mathbf{e}_k\|_2^2, \qquad \mathbf{q}_t \;=\; \mathbf{e}_{k_t}. \tag{3}$$

We use two techniques from VQ-VAE-2 [22] to manage the codebook. A commitment loss with a stop-gradient pulls the blob embeddings closer to their selected codes, while an entropy regularizer encourages diverse code usage to prevent *code collapse*. The code vectors are updated by an exponential moving average (EMA) during training, as shown in Appendix H. The codebook outputs the quantized substructure representations, forming $\mathbf{H}_{\text{q}} = [\mathbf{q}_1, \ldots, \mathbf{q}_T] \in \mathbb{R}^{T \times D}$.

2.5 Global–Blob Attention Fusion

Finally, we integrate the quantized blob embeddings with the global context vector using a single–query, multi–key attention [28], where $\mathbf{g}^{(T)}$ serves as the query and $\mathbf{H}_{\text{q}}$ provides keys and values. We first project the query, keys, and values, and compute attention weights over blobs:

$$\mathbf{q} = \mathbf{W}_q \, \mathbf{g}, \quad \mathbf{K} = \mathbf{W}_k \, \mathbf{H}_{\text{q}}, \quad \mathbf{V} = \mathbf{W}_v \, \mathbf{H}_{\text{q}}, \quad \boldsymbol{\pi} = \text{softmax}\Big(\tfrac{\mathbf{q}^\top \mathbf{K}}{\sqrt{D}}\Big). \tag{4}$$

The resulting weights $\boldsymbol{\pi}$ act as importance scores, which are used to compute a weighted blob summary $\mathbf{c}^*$. We then fuse this summary with the global vector $\mathbf{g}$ using a feature-wise gated residual connection. A small MLP computes a gate $\boldsymbol{\beta}$ that adaptively balances the contributions of the global and blob features. This final fused representation, $\mathbf{h}_{\text{fuse}}$, is then layer-normalized and passed to the classifier for prediction.

The full model is trained in two stages, proceeding from partitioning to focusing on quantization. See Appendix B for full training details, and Appendix D for time complexity analysis.

3 Experiments

Datasets We use benchmarks from ProteinShake [17]: **Gene Ontology (GO)** for multi-label molecular function prediction, **Enzyme Commission (EC)** for multi-class enzyme reaction classification, and **Structural Class (SCOP)** for multi-class fold family recognition. For each dataset, we use both a random split and a structure-based split. Dataset statistics are in Table 2.

Baselines and Implementations We benchmark **GNN**, a naïve **GVP-GNN**, and **GVP+DiffPool** against two stages of BIOBLOBS. GIN follows the ProteinShake baseline on residue graphs with a task-matched classifier. The naïve GVP-GNN uses the same GVP encoder as BIOBLOBS but applies graph-level pooling over residue embeddings, serving as a backbone baseline to show the added value of BIOBLOBS 's design. DiffPool provides a comparison to learned soft clustering: after the GVP encoder, a single DiffPool layer assigns residues to clusters, followed by cluster-level message passing and fusion with residue features for prediction. For BIOBLOBS, we fix hyperparameters across datasets and splits to highlight robustness, using the same protein structure encoder as GVP-GNN and setting the neural partitioner to at most $T = 5$ blobs of size $S = 15$ for efficiency and accuracy. We report results for Stage 1 (BIOBLOBS without codebook) and Stage 2 (BIOBLOBS with codebook) to isolate the contribution of each component. Hyperparameter details are in Appendix C.

Table 1: Experimental Results across three tasks, evaluated on both **random** and **structure-based** data splits. Shown are mean and standard deviation across four runs with different seeds.

Representation	Split	SCOP-FA	Enzyme Class	Gene Ontology
GNN	Random	0.495 ± 0.012	0.790 ± 0.007	0.704 ± 0.001
	Structure	0.415 ± 0.015	0.621 ± 0.026	0.474 ± 0.014
GVP-GNN	Random	0.4663 ± 0.001	0.801 ± 0.004	0.569 ± 0.032
	Structure	0.464 ± 0.027	0.451 ± 0.003	0.504 ± 0.004
GVP+DiffPool	Random	0.312 ± 0.013	0.807 ± 0.005	0.583 ± 0.001
	Structure	0.264 ± 0.022	0.657 ± 0.004	0.458 ± 0.016
BIOBLOBS w/o Codebook	Random	0.495 ± 0.009	0.823 ± 0.013	0.669 ± 0.001
	Structure	0.442 ± 0.001	0.671 ± 0.012	0.525 ± 0.001
BIOBLOBS	Random	$\mathbf{0.506 \pm 0.002}$	$\mathbf{0.840 \pm 0.001}$	$\mathbf{0.817 \pm 0.001}$
	Structure	$\mathbf{0.467 \pm 0.007}$	$\mathbf{0.684 \pm 0.004}$	$\mathbf{0.528 \pm 0.003}$

3.1 Protein Classification Performance

According to Tab. 1, BIOBLOBS achieves the best results on all three datasets under both *random* and *structure* splits, with larger margins on the structure split where similarity leakage is restricted. On EC (structure), the score rises from 0.621 for GNN to 0.684 with BIOBLOBS (**+10%**) and from 0.451 for GVP-GNN to 0.684 (**+52%**). Even without the codebook, BIOBLOBS surpasses GVP-GNN on EC and GO, showing that the neural partitioner and global–cluster attention fusion help the model capture function-relevant substructures instead of averaging over the full graph. Adding the codebook further improves accuracy without harming classification: the VQ-VAE's commitment loss compacts blob embeddings into reusable, function-related patterns, raising the GO (random) score from 0.669 to 0.817 (**+22%**). GVP-GNN itself does not reliably outperform GNN. On EC (structure), it falls to 0.451 compared to 0.621—likely because global pooling dilutes signals from small active sites. Combining DiffPool with GVP-GNN also fails, as one-shot soft coarsening merges distant or unrelated residues, which weakens functional signals and adds misleading features. By contrast, BIOBLOBS grows compact local blobs and fuses them with a light attention gate, reducing noise from irrelevant regions and aligning clusters with function labels.

3.2 BioBlobs Partition Interpretation

We offer initial analysis of the model's partitions by visualizing its blobs on proteins from the EC test set (structure split) using ChimeraX (Fig. 2). For structurally similar proteins like 1GBX and 1JUG, BIOBLOBS appears to consistently identify well-defined secondary structures in similar regions, such as α-helices and β-sheets, which shows its robustness in finding function-relevant substructures. The model's importance scores further improve this interpretability. High scores are assigned to coherent, stable structures while lower scores are given to disconnected regions, indicating that the

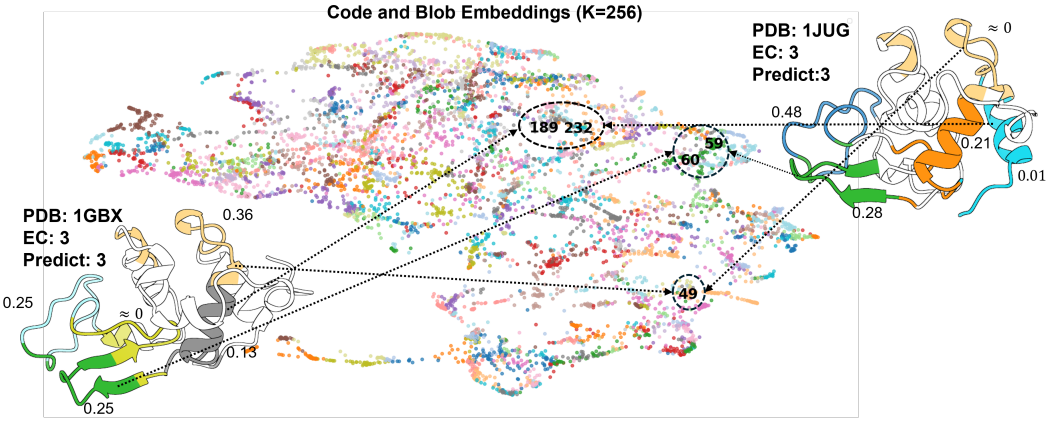

Figure 2: UMAP projection of the blob and code embeddings for the EC (structure) test set, where code embeddings are marked by their indices. Example BIOBLOBS partitions are shown on two sides. Colored regions represent distinct blobs and their code index. Each protein is annotated with its PDB ID, true and predicted EC numbers, and the importance score π_t for each blob.

model learns to prioritize key structural motifs. Finally, the codebook provides insight by mapping structurally similar blobs to the same or nearby codes. For example, a light orange motif in both proteins is quantized to code 49, while their green β-sheets map to the adjacent codes 59 and 60. Similarly, α-helices in both proteins map to the nearby codes 189 and 232. This demonstrates that the codebook learns a meaningful and consistent vocabulary for protein substructures. Therefore, we show that BIOBLOBS provides a robust and interpretable bridge between a protein's 3D structure and its learned, function-relevant substructures. The complete UMAP for blob and cluster embeddings is shown in Appendix F. This preliminary analysis suggests that future work should focus on systematic analysis of BIOBLOBS importance scores and properties of learned substructures.

4 Related Work

Protein Structure Modeling Protein GNNs typically build residue-level graphs (distance or k-NN) and pass messages; equivariant backbones such as GVP remain strong. Higher-order lifts via simplicial/cell-complex networks capture multi-body interactions [4, 13, 29], and surface-centric models learn directly on molecular surfaces from MaSIF onward [12].

Protein Motifs and Substructures Protein function is often mediated by recurrent modules such as catalytic triads, Rossmann cores, and P-loop NTPase sites [8, 24, 18]. Domain-scale taxonomies (SCOPe, CATH) catalog such units and are standard for supervision and leakage control [7, 21].

Graph Pooling, Subgraphs, and Partitioning Graph pooling coarsens graphs via soft assignments/selections (DiffPool, MinCutPool, Graph U-Nets/Top-K, SAGPool, ASAP) [19], while subgraph GNNs rely on rigid extraction [1]. In contrast, *partitioning* assigns each node to exactly one cluster under constraints; many formulations are NP-hard [11], motivating learned relaxations and neural combinatorial methods [16, 5].

Discrete Representation Learning Discrete latents turn continuous features into tokens: VQ-VAE with commitment/ST losses and its variants (VQ-VAE-2, VQGAN) connect to product/residual quantization [26, 22]. Protein-specific discretization (e.g., Foldseek's alphabet) shows the value of tokenized 3D patterns [27].

5 Conclusion

BIOBLOBS tackles the challenge of representing protein substructures of variable size and topology by introducing a differentiable seed–and–expand procedure for connected node sets, coupled with discrete codebook learning. Across three protein-property prediction benchmarks, BIOBLOBS combined with GVP-based feature extractors consistently outperforms strong baselines. Together, these results lay a foundation for more faithful protein representations and a scalable, systematic account of structure–function relationships.

References

[1] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *Advances in Neural Information Processing Systems*, 33:8017–8029, 2020.

[2] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000. doi: 10.1038/75556.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Lio, Guido F Montufar, and Michael Bronstein. Weisfeiler and lehman go cellular: Cw networks. *Advances in neural information processing systems*, 34:2625–2640, 2021.

[5] Giorgos Bouritsas, Andreas Loukas, Nikolaos Karalias, and Michael Bronstein. Partition and code: learning how to compress graphs. *Advances in Neural Information Processing Systems*, 34:18603–18619, 2021.

[6] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. *Algorithm engineering: selected results and surveys*, pages 117–158, 2016.

[7] John-Marc Chandonia, Lindsey Guan, Shiangyi Lin, Changhua Yu, Naomi K Fox, and Steven E Brenner. Scope: improvements to the structural classification of proteins–extended database to facilitate variant interpretation and machine learning. *Nucleic acids research*, 50(D1): D553–D559, 2022.

[8] G. Dodson and A. Wlodawer. Catalytic triads and their relatives. *Trends in Biochemical Sciences*, 23(9):347–352, 1998. doi: 10.1016/S0968-0004(98)01254-7.

[9] Alexandre Duval, Simon V Mathis, Chaitanya K Joshi, Victor Schmidt, Santiago Miret, Fragkiskos D Malliaros, Taco Cohen, Pietro Lio, Yoshua Bengio, and Michael Bronstein. A hitchhiker's guide to geometric gnns for 3d atomic systems. *arXiv preprint arXiv:2312.07511*, 2023.

[10] Romanos Fasoulis, Georgios Paliouras, and Lydia E Kavraki. Graph representation learning for structural proteomics. *Emerging Topics in Life Sciences*, 5(6):789–802, 2021.

[11] Tomas Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. Complexity of graph partition problems. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 464–472, 1999.

[12] Pablo Gainza, Freyr Sverrisson, Frederico Monti, Emanuele Rodola, Davide Boscaini, Michael M Bronstein, and Bruno E Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature methods*, 17(2):184–192, 2020.

[13] Christopher Wei Jin Goh, Cristian Bodnar, and Pietro Lio. Simplicial attention networks. *arXiv preprint arXiv:2204.09455*, 2022.

[14] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[15] Bowen Jing, Stephan Eismann, Patricia Suriana, Raphael J. L. Townshend, and Ron Dror. Learning from Protein Structure with Geometric Vector Perceptrons, May 2021. URL `http://arxiv.org/abs/2009.01411`. arXiv:2009.01411 [q-bio].

[16] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672, 2020.

[17] Tim Kucera, Carlos Oliver, Dexiong Chen, and Karsten Borgwardt. Proteinshake: Building datasets and benchmarks for deep learning on protein structures. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) — Datasets and Benchmarks Track*, pages 58277–58289, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/b6167294ed3d6fc61e11e1592ce5cb77-Paper-Datasets_and_Benchmarks.pdf.

[18] Dmitry D. Leipe, Eugene V. Koonin, and L. Aravind. Evolution and classification of p-loop kinases and NTPases. *Journal of Molecular Biology*, 333(4):781–815, 2003. doi: 10.1016/j.jmb.2003.08.031.

[19] Chuang Liu, Yibing Zhan, Jia Wu, Chang Li, Bo Du, Wenbin Hu, Tongliang Liu, and Dacheng Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities. *arXiv preprint arXiv:2204.07321*, 2022.

[20] Loredana Lo Conte, Bart Ailey, Tim JP Hubbard, Steven E Brenner, Alexey G Murzin, and Cyrus Chothia. Scop: a structural classification of proteins database. *Nucleic acids research*, 28 (1):257–259, 2000.

[21] Christine A Orengo, Alex D Michie, Susan Jones, David T Jones, Mark B Swindells, and Janet M Thornton. Cath–a hierarchic classification of protein domain structures. *Structure*, 5 (8):1093–1109, 1997.

[22] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

[23] Mary M Rorick and Günter P Wagner. Protein structural modularity and robustness are associated with evolvability. *Genome biology and evolution*, 3:456–475, 2011.

[24] Michael G. Rossmann, Dino Moras, and Kenneth W. Olsen. Chemical and biological evolution of a nucleotide-binding protein. *Nature*, 250:194–199, 1974. doi: 10.1038/250194a0.

[25] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32(Database issue):D431–D433, 2004. doi: 10.1093/nar/gkh070.

[26] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[27] Michel Van Kempen, Stephanie S Kim, Charlotte Tumescheit, Milot Mirdita, Jeongjae Lee, Cameron LM Gilchrist, Johannes Söding, and Martin Steinegger. Fast and accurate protein structure search with foldseek. *Nature biotechnology*, 42(2):243–246, 2024.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[29] Zhiyu Wang, Arian Jamasb, Mustafa Hajij, Alex Morehead, Luke Braithwaite, and Pietro Liò. Topotein: Topological deep learning for protein representation learning. *arXiv preprint arXiv:2509.03885*, 2025.

Use of Large Language Models (LLMs)

LLMs were used to assist in coding, writing, and producing figures. All LLM-produced code and text was thoroughly double-checked.

A Reproducibility

All code, data, and weights necessary to reproduce results and use our models on new data are available on `https://anonymous.4open.science/r/BioBlobs-EECD/`. Benchmarking was done with the ProteinShake [17] open-source library to ensure reproducibility.

B Training Details

B.1 Multi-Stage Training

We train BIOBLOBS using a stable two-stage process. In **Stage 1**, we train the encoder, partitioner, and attention fusion module using only the cross-entropy classification loss. This allows the model to first learn how to partition proteins into meaningful blobs without the added complexity of quantization. In **Stage 2**, we initialize the substructure codebook by applying K–means clustering to the blob embeddings learned in the first stage, which provides a strong starting point for the codes. We then fine-tune the entire model end-to-end, introducing commitment and entropy losses to optimize the codebook alongside the classification objective. To ensure stable convergence, the weights of these new losses are gradually increased. This progressive approach allows the model to learn an effective partitioning scheme before learning the discrete substructure vocabulary, which reduces training instability.

C Model Hyperparameters

Default Hyperparameters. We summarize the default hyperparameters used in the two–stage training of BIOBLOBS across 3 datasets and 2 splits.

Stage 1 (baseline training). We train the encoder, partitioner, and global–cluster attention fusion for 120 epochs with batch size 128 and learning rate 10^{-3}. The GVP encoder uses scalar–vector input dimensions of $(6, 3)$ for residues and $(32, 1)$ for edges, with hidden dimensions $(100, 16)$ and $(32, 1)$, respectively. We stack three GVP convolution layers with dropout rate 0.1 and apply sum pooling for graph readout. The neural partitioner is configured with up to $T = 5$ clusters, hidden dimension 50, $k = 1$ hop expansion, maximum cluster size 15, termination threshold 0.95, and temperature annealing from $\tau_{\text{init}} = 1.0$ to $\tau_{\min} = 0.1$ with exponential decay 0.95.

Stage 2 (joint training with codebook). We resume from the Stage 1 checkpoint and initialize the substructure codebook using K–means clustering on the pre-trained cluster embeddings. The codebook has 256 entries of dimension 100 (matching the scalar hidden dimension) and is updated by EMA with decay 0.99. Training continues for 30 epochs with reduced learning rate 10^{-4} and without backbone freezing. Additional objectives are introduced: the commitment loss with weight $\lambda_{\text{vq}} = 1.0$ and the entropy regularizer with $\lambda_{\text{ent}} = 0.1$. This staged procedure allows the encoder and partitioner to first stabilize on meaningful blobs, after which the codebook is optimized jointly without destabilizing the backbone.

D Time Complexity Analysis

The full pipeline after the protein encoder runs in near-linear time with respect to the number of residues. Among the components, the Neural Partitioner contributes the main cost, while the substructure codebook and global–blob attention add only modest overhead.

Neural Partitioner. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote the protein graph with $N = |\mathcal{V}|$ residues and $|\mathcal{E}| = O(k_{\text{NN}} N)$ edges, where k_{NN} is the number of neighbors used in the k-nearest neighbor construction of the graph. The partitioner builds up to T blobs through the iterative seed–threshold–expansion

process, where k_{hop} controls the radius of the neighborhood explored around each seed. At each iteration, three steps dominate the cost:

(i) *Seed selection:* the scorer computes logits over all N residues and applies Gumbel–Softmax sampling, which is linear in the number of nodes, $O(N)$.

(ii) *Neighborhood construction:* the k_{hop}-hop neighborhood of the seed is obtained through a sparse traversal, which touches up to k_{hop} layers of edges and costs $O(k_{\text{hop}}|\mathcal{E}|)$.

(iii) *Blob expansion:* each candidate in the neighborhood is scored by a seed-conditioned dot product, converted into probabilities, and thresholded, which again requires linear work in the number of candidates, bounded by $O(N)$.

Thus, the per-iteration complexity is $O(k_{\text{hop}}|\mathcal{E}| + N)$, and over T iterations the total becomes

$$O\left(T\left(k_{\text{hop}}\,|\mathcal{E}| + N\right)\right) = O\left(T\left(k_{\text{hop}}\,k_{\text{NN}} + 1\right)N\right). \tag{5}$$

An additional $O(T^2N)$ term arises from non-overlap checks across blobs, but this remains minor when T is small.

Substructure Codebook. For each of the T blob embeddings with dimension D, the codebook computes squared distances to all K code vectors. This requires $O(TKD)$ time, followed by nearest-neighbor assignment and an EMA update of the selected codes. Both operations share the same order of complexity, so quantization remains linear in the number of blobs and codes.

Global–Blob Attention. The fusion step integrates the global protein context with the T blob embeddings. Projecting queries, keys, and values has cost $O((N+T)D^2)$, but since these projections are reused and D is fixed, the dominant term is the attention operation itself. Computing attention scores across T blobs with H heads requires $O(THD)$, followed by weighted pooling and a feature-wise gating step in $O(D^2)$. Because $T \ll N$ and H is fixed, the fusion module is lightweight compared to the partitioner.

E Benchmark Dataset Info

We benchmark our method on Gene Ontology (GO) [2] for protein functions, Enzyme Commission (EC) [25] for catalytic reactions, and SCOP [7, 20] for structural classification, with dataset statistics shown in Table 2.

Table 2: Details on Benchmark Datasets

Dataset	**# Proteins**	**Area of protein biology**	**Metric**
Gene Ontology (GO)	32,633	Functions, Components, Pathways	F_{max}
Enzyme Commission (EC)	15,603	Reaction catalysis	Accuracy
Structural class (SCOP)	10,066	Geometric relationships	Accuracy

F Codebook and Blob Embedding Visualization

We obtain blob embeddings before quantization and compare them with code embeddings extracted from the codebook. Using UMAP, we visualize both on the EC test set under random and structure splits, where red integers mark the positions of code embeddings. In the structure split, code embeddings appear more densely packed and form clearer clusters compared to the random split. This aligns with the fact that the structure split places structurally similar proteins (0.7) in the test set.

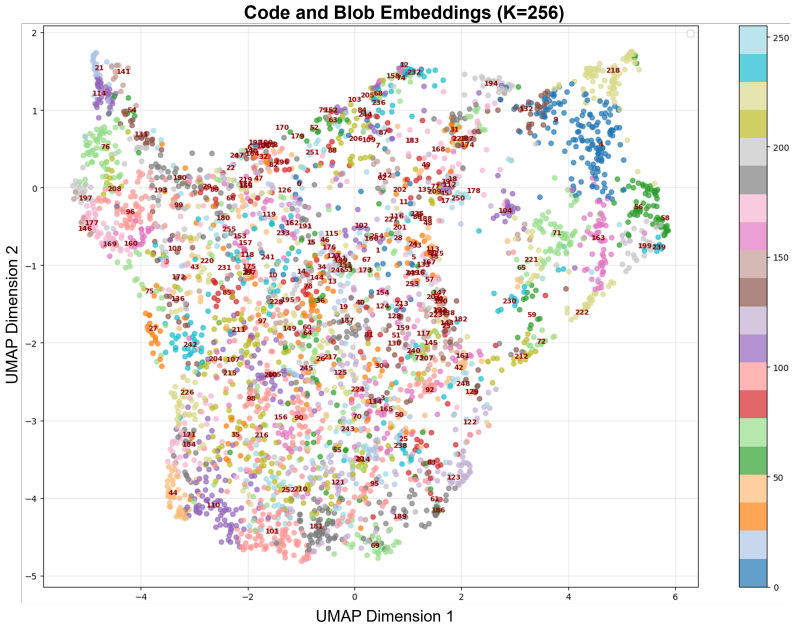

Figure 3: UMAP projection of code and cluster embeddings for EC dataset, random split

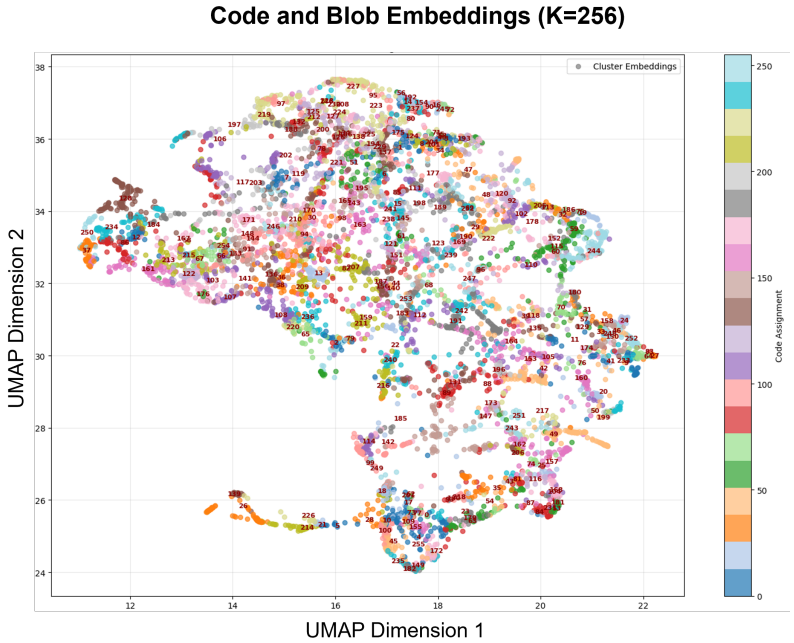

Figure 4: UMAP projection of code and cluster embeddings for EC dataset, structure split

G Neural Blob Partitioner Algorithm

The full algorithm of the partitioner is displayed below.

Algorithm 1 Neural BLOB Partitioner

Require: Residue embeddings $\mathbf{Z} \in \mathbb{R}^{N \times D}$, edge index $\mathcal{E}$, max blobs T, max blob size S, k-hop radius k, temperature schedule $\{\tau^{(t)}\}$, termination threshold ρ, step size α_g

Ensure: Assignment $\mathbf{M} \in \{0,1\}^{N \times T}$, blob embeddings $\mathbf{H} = \big[\mathbf{h}_{\text{clus}}^{(1)}, \ldots, \mathbf{h}_{\text{clus}}^{(T)}\big]$

$\mathbf{g}^{(0)} \leftarrow \text{Pool}(\mathbf{Z})$; $\mathbf{M} \leftarrow \mathbf{0}$; $\mathbf{m} \leftarrow \mathbf{1}_N$ ▷ global context, blob assignment, residue availability

for $t = 1$ to T **do**

  Early stop: if $\text{coverage}(\mathbf{M}) \geq \rho$ or no i with $m_i = 1$, **break**

  $\boldsymbol{\ell}^{(t)} \leftarrow f_{\text{seed}}\big([\mathbf{Z}; \mathbf{g}^{(t-1)}]\big)$; mask unavailable: $\tilde{\boldsymbol{\ell}}^{(t)} = \boldsymbol{\ell}^{(t)} + \log \mathbf{m}$

  $\mathbf{w}^{(t)} \leftarrow \text{GumbelSoftmax}\big(\tilde{\boldsymbol{\ell}}^{(t)}, \tau^{(t)}\big)$ ▷ hard variant with straight-through (ST)

  $\mathbf{e}^{(t)} \leftarrow \text{one\_hot}(\arg\max_i w_i^{(t)})$; $\tilde{\mathbf{w}}^{(t)} \leftarrow \mathbf{e}^{(t)} + (\mathbf{w}^{(t)} - \text{sg}(\mathbf{w}^{(t)}))$

  $\mathbf{z}_{\text{seed}}^{\text{ST}} \leftarrow \sum_i \tilde{w}_i^{(t)} \mathbf{z}_i$; $i_s \leftarrow \arg\max_i e_i^{(t)}$

  $\mathbf{M}[i_s, t] \leftarrow 1$; $m_{i_s} \leftarrow 0$

  Candidate set and local stats

  $\mathcal{R}^{(t)} \leftarrow \mathcal{N}_k(i_s; \mathcal{E}) \cap \{i : m_i = 1\}$ ▷ k-hop and available

  $\boldsymbol{\phi}^{(t)} \leftarrow \text{LocalStats}(\mathcal{R}^{(t)})$ ▷ log#cands, seed-link fraction, induced density

  Threshold prediction

  $\theta^{(t)} \leftarrow f_{\text{thresh}}\big([\mathbf{z}_{\text{seed}}^{\text{ST}}; \mathbf{g}^{(t-1)}; \boldsymbol{\phi}^{(t)}]\big)$

  Blob expansion

  For $i \in \mathcal{R}^{(t)}$: $s_i^{(t)} = \frac{(\mathbf{W}_k \mathbf{z}_i)^\top (\mathbf{W}_q \mathbf{z}_{\text{seed}}^{\text{ST}})}{\sqrt{d}}$ ▷ optionally add normalized link term

  $p_i^{(t)} = \sigma\big((s_i^{(t)} - \theta^{(t)})/\tau^{(t)}\big)$; $c_i^{(t)} = \mathbb{I}[p_i^{(t)} \geq 0.5]$ ▷ ST hard gate

  Keep the seed and the top-S residues by $p_i^{(t)}$; set other $c_i^{(t)} = 0$

  For all i: $\mathbf{M}[i, t] \leftarrow c_i^{(t)}$; $m_i \leftarrow m_i \cdot (1 - c_i^{(t)})$

  Blob embedding (ST) and context update

  $\mathbf{h}_{\text{hard}}^{(t)} = \frac{\sum_i c_i^{(t)} \mathbf{z}_i}{\sum_i c_i^{(t)} + \varepsilon}$, $\mathbf{h}_{\text{soft}}^{(t)} = \frac{\sum_i \tilde{p}_i^{(t)} \mathbf{z}_i}{\sum_i \tilde{p}_i^{(t)} + \varepsilon}$, $\mathbf{h}_{\text{clus}}^{(t)} = \mathbf{h}_{\text{hard}}^{(t)} + \big(\mathbf{h}_{\text{soft}}^{(t)} - \text{sg}(\mathbf{h}_{\text{soft}}^{(t)})\big)$

  $\mathbf{g}^{(t)} \leftarrow \mathbf{g}^{(t-1)} + \alpha_g \, f_{\text{global}}(\mathbf{h}_{\text{clus}}^{(t)})$

end for

return $\mathbf{M}$, $\mathbf{H}$

H EMA update of the codebook

We maintain a vector-quantization codebook $\mathbf{E} = \{\mathbf{e}_k\}_{k=1}^{K}$ and update it with exponential moving averages (EMA) as in VQ-VAE [26]. Let $\mathcal{H} = \{\mathbf{h}_s\}$ be the set of cluster embeddings in a minibatch, and let $r_{s,k} \in \{0, 1\}$ denote the hard assignment of $\mathbf{h}_s$ to code k (or soft responsibilities in a soft variant). Define the batch counts and sums

$$n_k = \sum_s r_{s,k}, \qquad \mathbf{u}_k = \sum_s r_{s,k}\, \mathbf{h}_s.$$

With decay $\lambda \in (0, 1)$, EMA keeps running totals $\hat{N}_k$ and $\hat{\mathbf{U}}_k$:

$$\hat{N}_k^{(t)} = \lambda\, \hat{N}_k^{(t-1)} + (1 - \lambda)\, n_k, \qquad \hat{\mathbf{U}}_k^{(t)} = \lambda\, \hat{\mathbf{U}}_k^{(t-1)} + (1 - \lambda)\, \mathbf{u}_k,$$

and updates each code vector to the normalized running mean

$$\mathbf{e}_k^{(t)} = \frac{\hat{\mathbf{U}}_k^{(t)}}{\hat{N}_k^{(t)} + \varepsilon}.$$

We apply stop-gradient to $\hat{N}_k^{(t)}$ and $\hat{\mathbf{U}}_k^{(t)}$ so that code updates do not backpropagate into the encoder. If the accumulators start at zero, an optional bias correction divides numerator and denominator by $(1 - \lambda^t)$. A small ε prevents division by zero and stabilizes the update when counts are low. In practice, $\lambda \in [0.99, 0.999]$ works well; codes with near-zero effective count can be reinitialized to recent cluster embeddings to avoid dead entries. The EMA path replaces direct gradient descent on $\mathbf{E}$ and tracks the empirical means of assigned embeddings with lower variance when assignments fluctuate.

I Neural Partitioner Case Study

In Fig. 5, we study how the neural partitioner parameters influence the test performance of BIOBLOBS on the EC and SCOP-FA datasets. We focus on two factors: the maximum blob size S and the maximum number of blobs T, while keeping other hyperparameters fixed. For maximum blob size, we fix $T = 5$. The left figure shows that increasing S from 5 to 15 gives slight performance improvements, but pushing it further to 25 causes a clear drop. This suggests that both datasets benefit from substructures of moderate size (5–10 residues), whereas overly large blobs tend to merge unrelated residues, thereby diluting function-specific signals and reducing accuracy. For the number of blobs, we fix $S = 15$. The right figure shows a similar pattern across datasets. Accuracy peaks when only five blobs are used, then declines as more blobs are introduced, likely because additional, function-irrelevant blobs compete for attention and confuse the classifier. Interestingly, when $T = 25$, the performance rebounds. At this point, nearly all residues are assigned to blobs, which raises the chance of covering function-relevant substructures and improves prediction. Overall, both "how many" and "how large" the blobs are should accommodate the specific protein predictive task. When labels depend on short, localized motifs (for example, catalytic or binding patches), a maximum blob size of 10–15 is effective. When labels depend on domain-level organization, a larger cap around 25 may help.

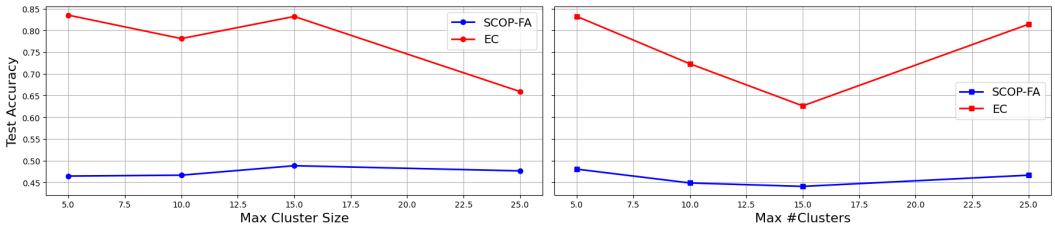

Figure 5: Neural partitioner case study: tuning maximum cluster size S and number of clusters T.