Supplemental Material for *Deep Generative Models of Protein Structure: A Method to Uncover Distant Relationships Across a Continuous Fold Space*

Eli J. Draizen[1,2*], Stella Veretnik[2], Cameron Mura[1,2*], and Philip E. Bourne[1,2]

[1]Department of Biomedical Engineering,

[2]School of Data Science, University of Virginia, Charlottesville, VA, USA

[*]Corresponding Authors: E-mail: e.draizen@gmail.com and cmura@virginia.edu

doi:xx.xxxx/xxxx.x.xx.xxxxxxx

April 18, 2023

Contents

Table 1: CATH Superfamilies used in this study.

CATH Code	Name	Description	# Domains	# Representatives	Manual Urfold
1.10.10.10	Winged helix-like DNA-binding domain superfamily/Winged helix DNA-binding domain		3444	524	
1.10.238.10	EF-hand		1933	166	
1.10.490.10	Globins		2891	52	
1.10.510.10	Transferase (Phosphotransferase) domain 1		7219	148	
1.20.1260.10	Ferritin, core subunit, four-helix bundle	a major non-haem iron storage protein in animal, plants and microorganisms	2985	60	
2.30.30.100	SH3 type barrels	Includes Integrase, C-terminal domain superfamily, retroviral	1545	56	SBB
2.40.50.140	OB fold	Dihydrolipoamide Acetyltransferase, E2P Nucleic acid-binding proteins"	2879	227	SBB
2.60.40.10	Immunoglobulin		31905	873	
3.10.20.30	Beta-grasp domain	a core structure consisting of beta(2)-alpha-beta(2), which is similar to that found in ubiquitin. Domains with this type of structure are found in the 2Fe-2S ferredoxin family	520	48	beta-grasp (Ub)
3.30.1360.40	Gyrase A; domain 2		160	13	Sm-like ribonucleoproteins
3.30.1370.10	''K Homology domain, type 1"		139	34	RRM/RBD(ish)
3.30.1380.10	Hedgehog domain		101	10	Sm-like ribonucleoproteins
3.30.230.10	Ribosomal Protein S5; domain 2		1274	45	Sm-like ribonucleoproteins
3.30.300.20	K homology (KH) domain	Could belong together w/ the RRMs in a new Urfold	529	30	RRM/RBD(ish)
3.30.310.60	Sm-like ribonucleoprotein	C-terminal domain	28	1	Sm-like ribonucleoproteins
3.40.50.300	P-loop NTPases	P-loop containing nucleotide triphosphate hydrolases (both 2ak3 and 1reb in same cath id)	9233	561	P-loop NTPases
3.40.50.720	NAD(P)-binding Rossmann-like Domain		11728	647	Rossmann-based
3.80.10.10	Ribonuclease Inhibitor	Luecine rich repeat protein; positive control, sanity check	709	99	
3.90.420.10	''Oxidoreductase, molybdopterin-binding domain"		58	6	beta-grasp (Ub)
3.90.79.10	Nucleoside Triphosphate Pyrophosphohydrolase		850	74	beta-grasp (Ub)

2 Voxelization & Featurization

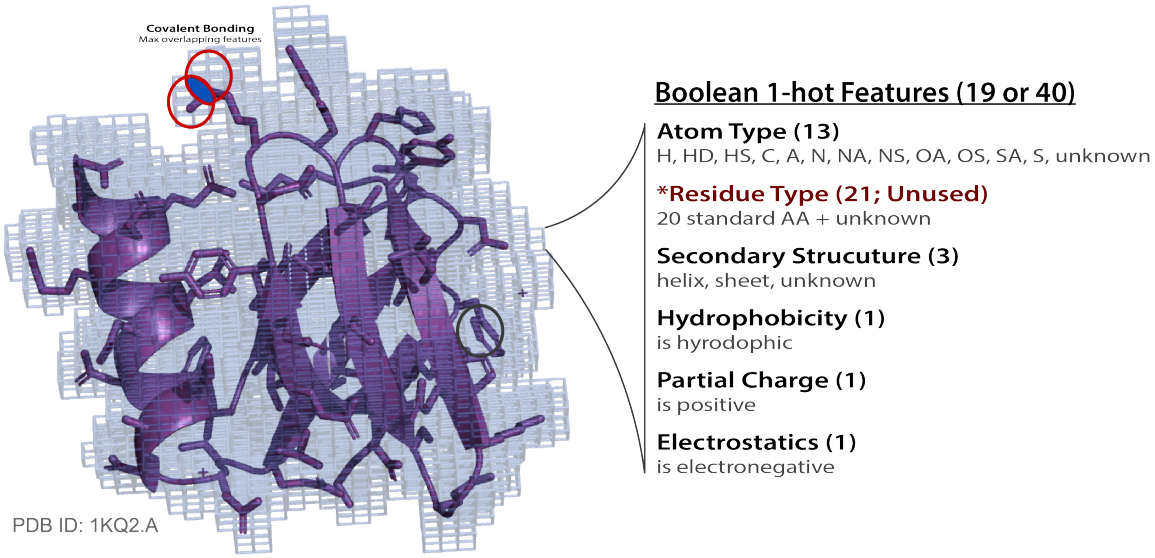


Figure 1: Each domain is voxelized by (1) centering in a 256 $Å^3$ volume; and (2) discretizing each atom to fit 1 ^{3} voxels searching a KD-Tree with a radius equal to the current atoms van der walls radii, where the KD-tree initialized by the full 256 $Å^3$ volume, with 1 ^{3} resolution. If two or more atoms occupy a single voxel, the maximum from each feature is used to handle covalent bonding. Each voxel contains a 1-hot feature vector with 19 or 40 features depending on if residue type is included. Residue type was not included in the models shown in this paper because due poor reconstruction metrics so was not considered useful for this type of model, seen in Fig 3

3 Immunoglobulin (2.60.40.10) Model Metrics

3.1 Training & Validation Loss

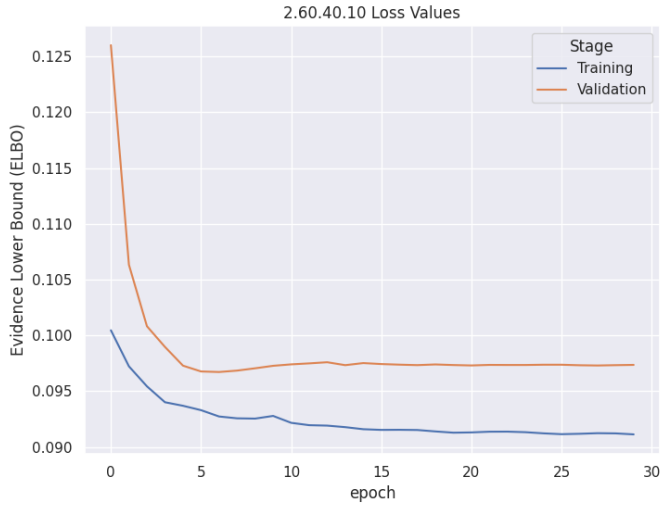


Figure 2: The 2.60.40.10 model was trained for 30 epochs using a 80% / 10 % split from CATH's S35 clusters (test [10 %] not shown).

3.2 Classification Metrics (w/ Residue Type)

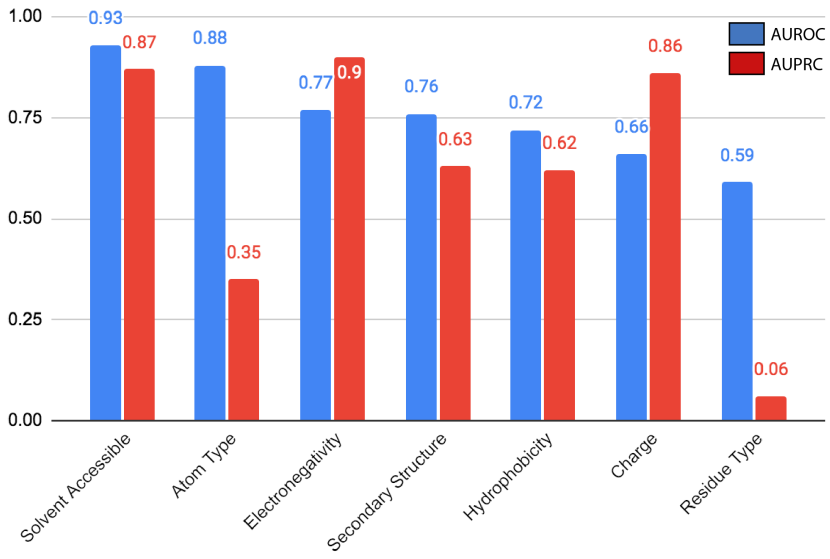


Figure 3: We trained an Immunoglobulin-specific model 7 different times for 7 different groups of features, excluding all of the other feature groups. We compared the reconstructed values for each values to the input using ROC (receiver operating; true positive vs false positive) and PRC (precision vs recall) curves, saving the AUC (area under curve) for each. Most features were able to be reconstructed well (AUC ≥ 0.6) except residue type so we removed them from further models. We hypothesize that residue type is not as important for atom-only models and is too coarse-grained to be meaningful for this context.

3.3 Classification Metrics (w/o Residue Type)

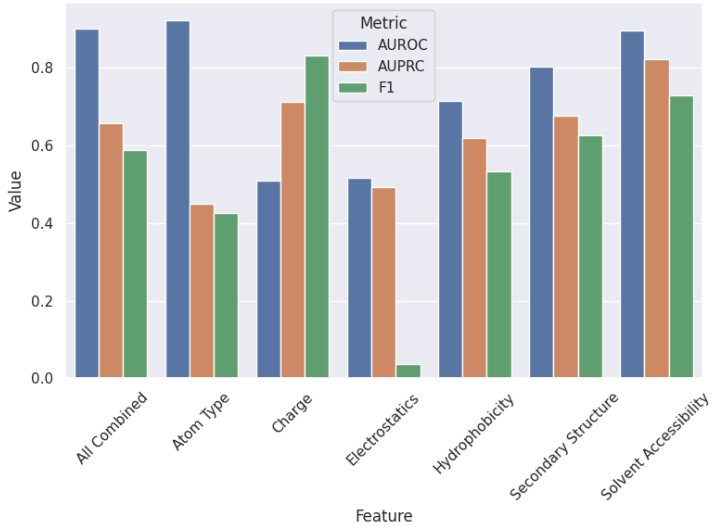
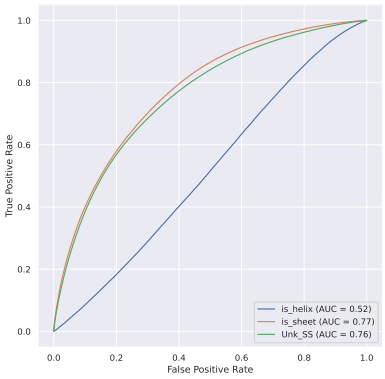
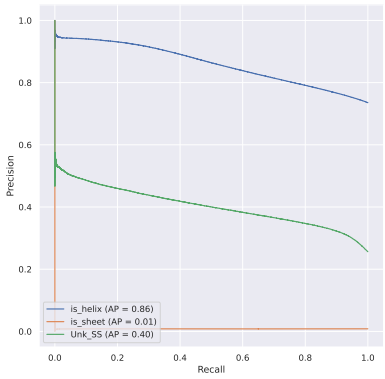


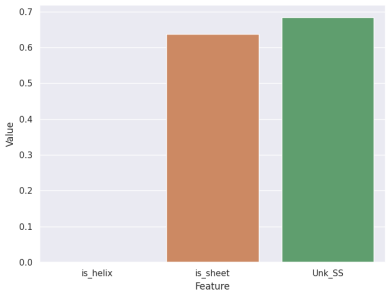
Figure 4: The 2.60.40.10 model was trained with all features, but individual feature groups were separated to perform micro-averaging ROC, PRC, and F1 scores



(a) Micro-averaged ROC Curve for separated features in the Secondary Structure feature group.

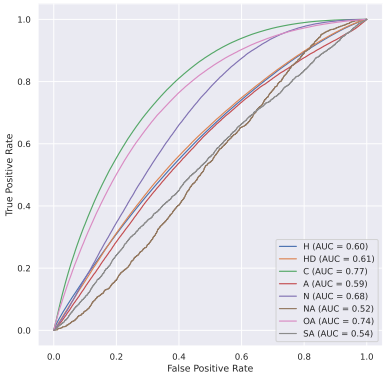


(b) Micro-averaged PRC Curve for separated features in the Secondary Structure feature group.

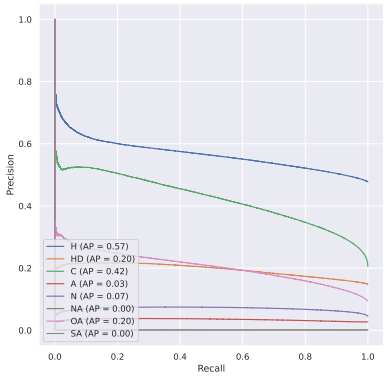


(c) Micro-averaged F1 values for separated features in the Secondary Structure feature group.

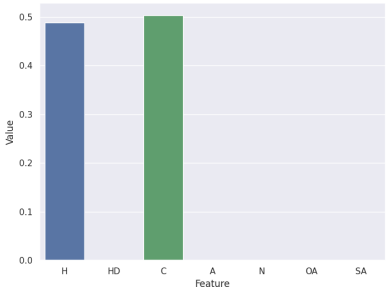
Figure 5: Classification metrics for separated features in the Secondary Structure feature group.



(a) Micro-averaged ROC Curve for separated features in the Atom Type feature group.



(b) Micro-averaged PRC Curve for separated features in the Atom Type feature group.

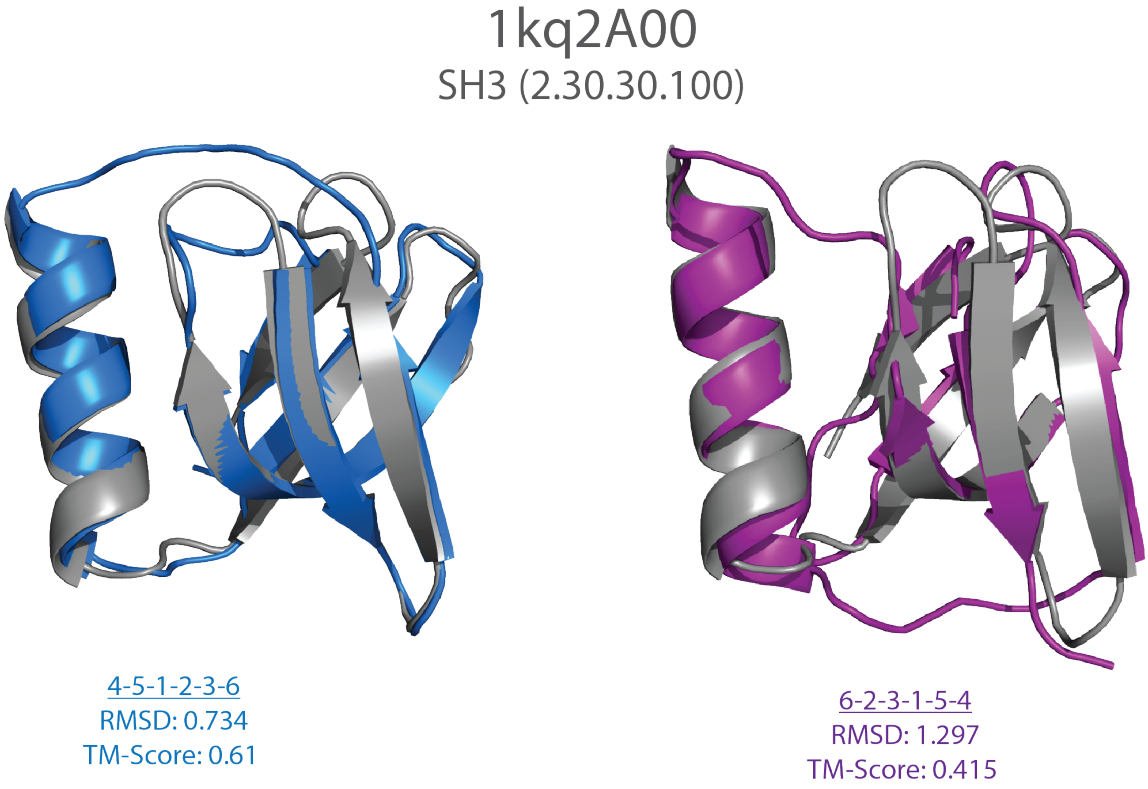


(c) Micro-averaged F1 values for separated features in the Atom Type feature group.

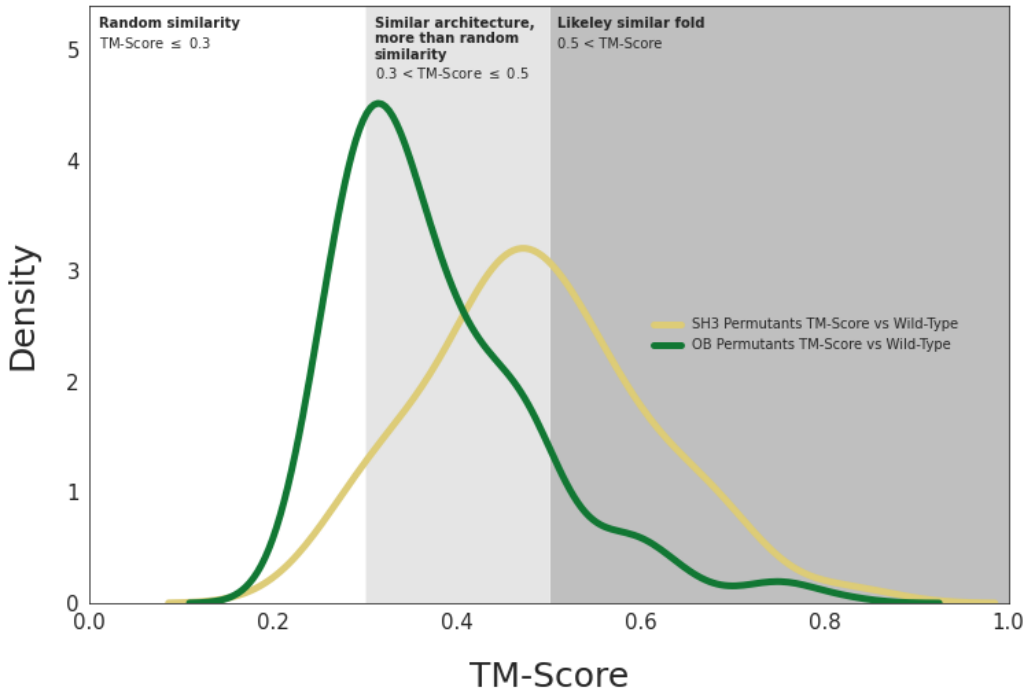
Figure 6: Classification metrics for separated features in the Atom Type feature group.

4 Multiple Loop Permutations

4.1 Permutants



(a) Exemplar SH3 permutatants



(b) TM-Scores for Multiple Loop Permuted structures. Many of the SH3 and OB permutant TM-Scores fall between 0.3-0.5, showing that that are more than randomly similar, but not the same fold.

Figure 7: Multiple Loop Permutations Example

4.2 Class Imbalance Scores

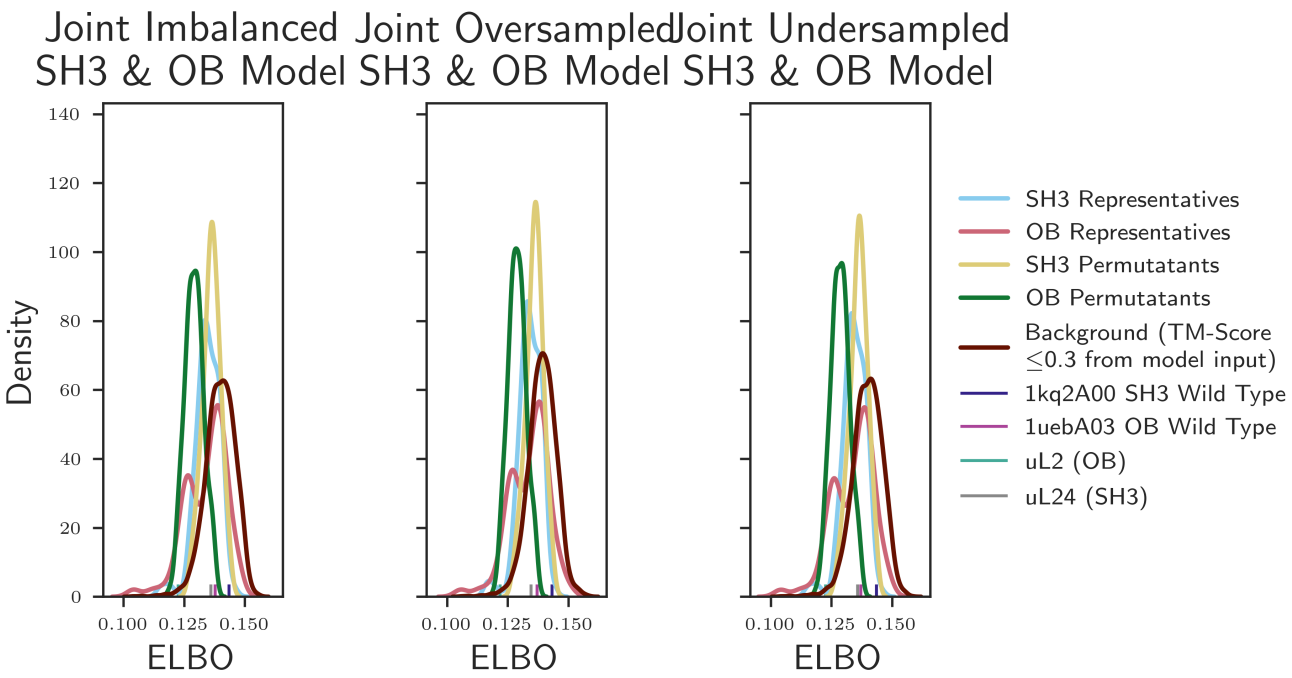
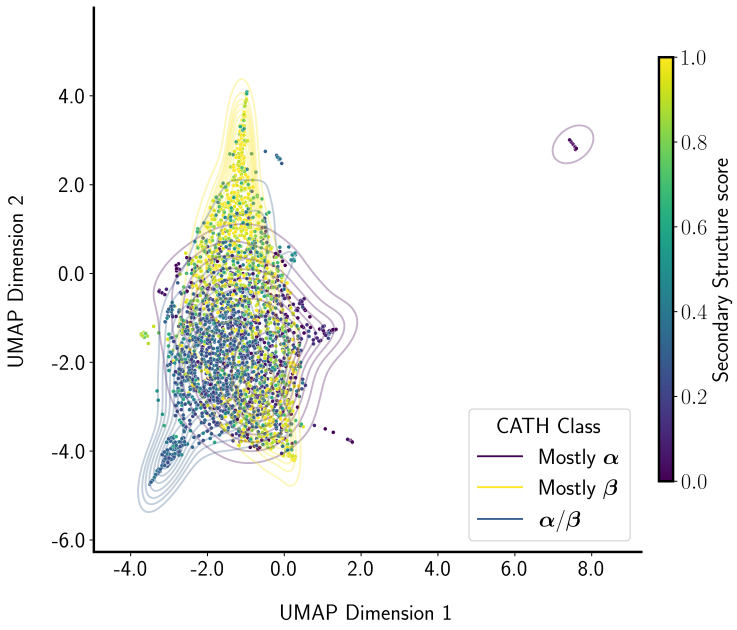


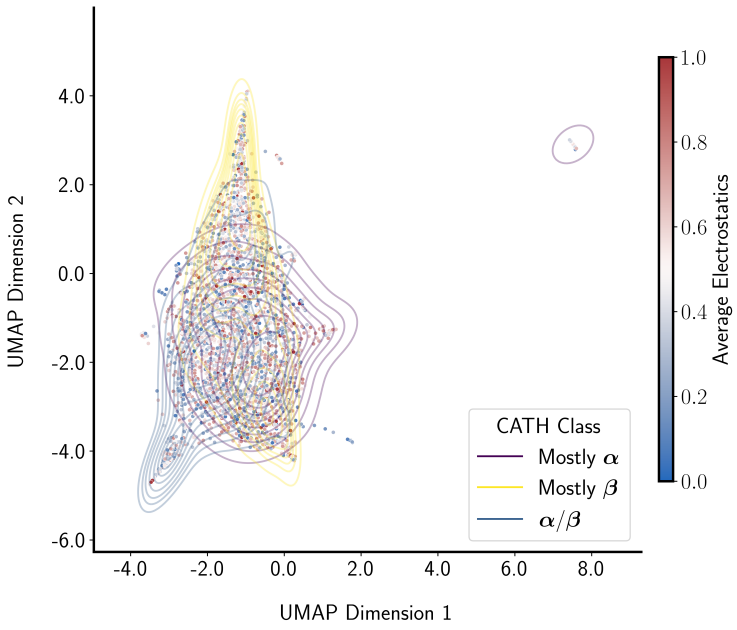
Figure 8: In order test how class imbalance affects our models, we trained 3 joint SH3 and OB models: (A) using all domains from each superfamily; (B) oversampling SH3 domains to match the number of OB domains; and (3) under-sampling OB domains to match the number of SH3 domains. We found no significant change between them in terms of ELBO scores when running representatives, multi-loop permuted models, and ancestral versions of SH3 and OB.

5 Latent Space

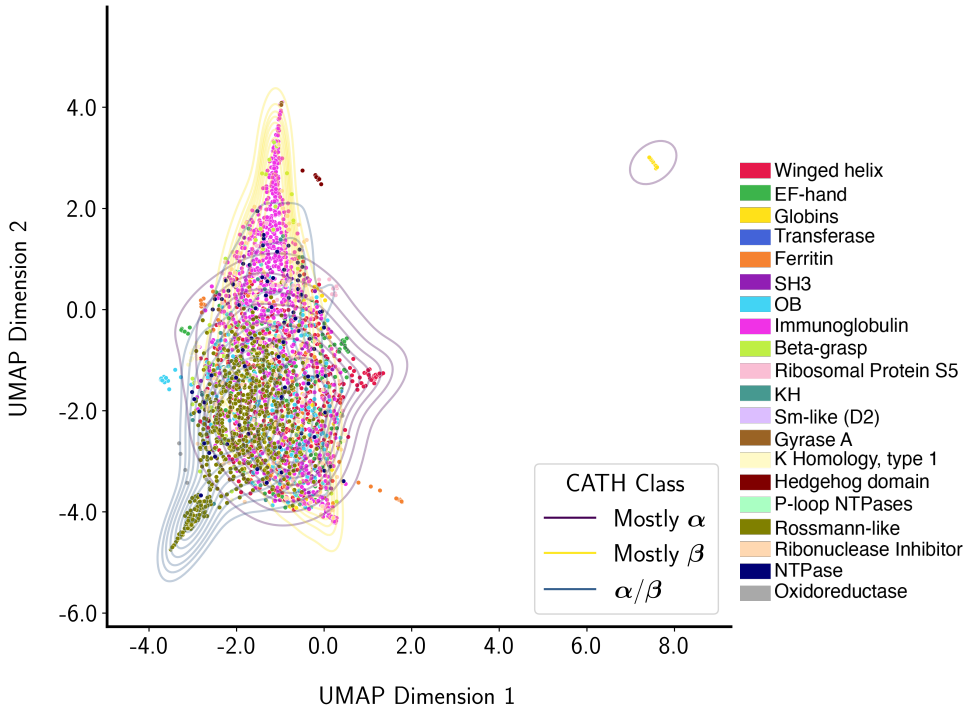
5.1 UMAP



(a) Colored by secondary structure



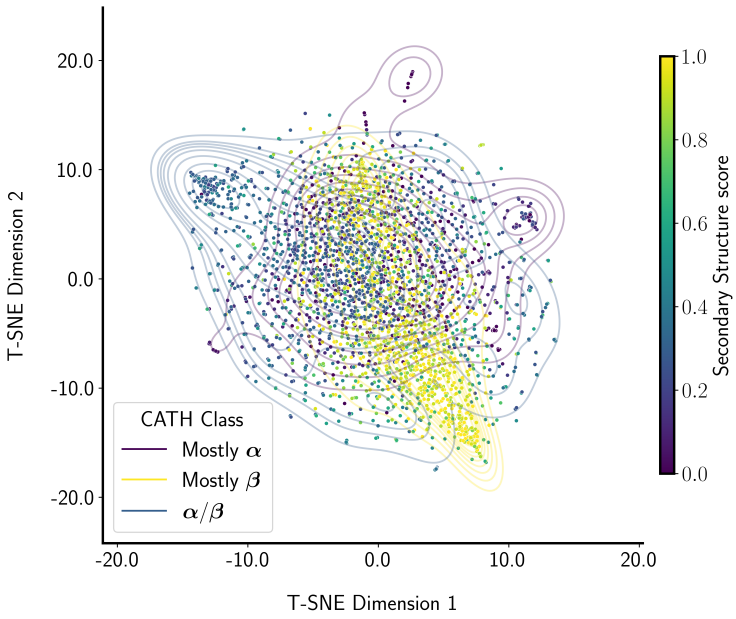
(b) Colored by averaged electro-negativity values



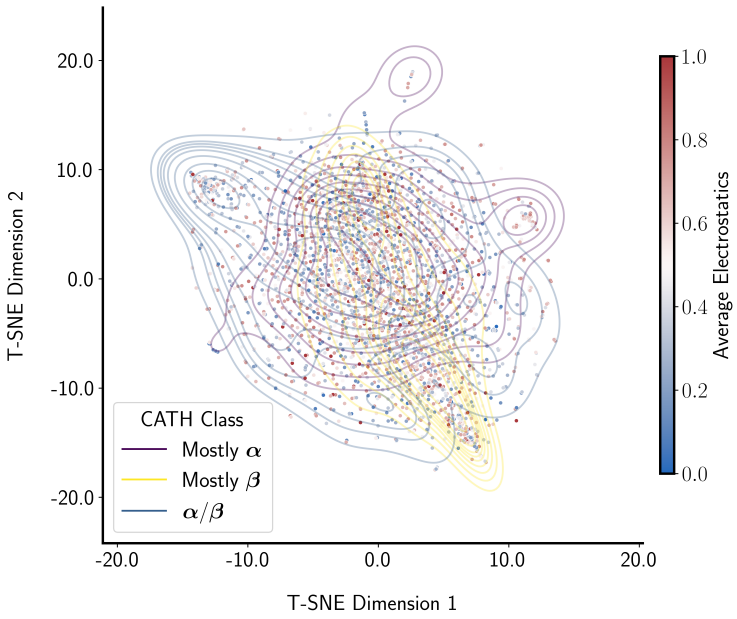
(c) Colored by true CATH Superfamily

Figure 9: Latent Space from UMAP. Representatives from each superfamily were subjected to models trained with domains from the same superfamily, saving the latent variable representing the mean for each representative domain. The latent variables for each different model were concatenated and reduced from 1024 dims to 2 for visualization

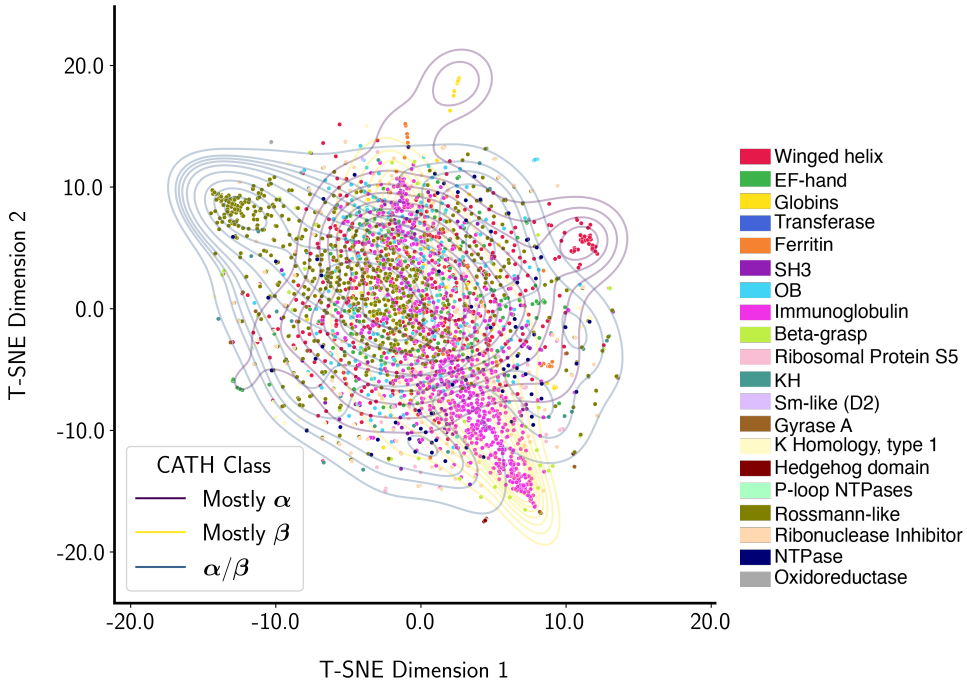
5.2 t-SNE



(a) Colored by secondary structure



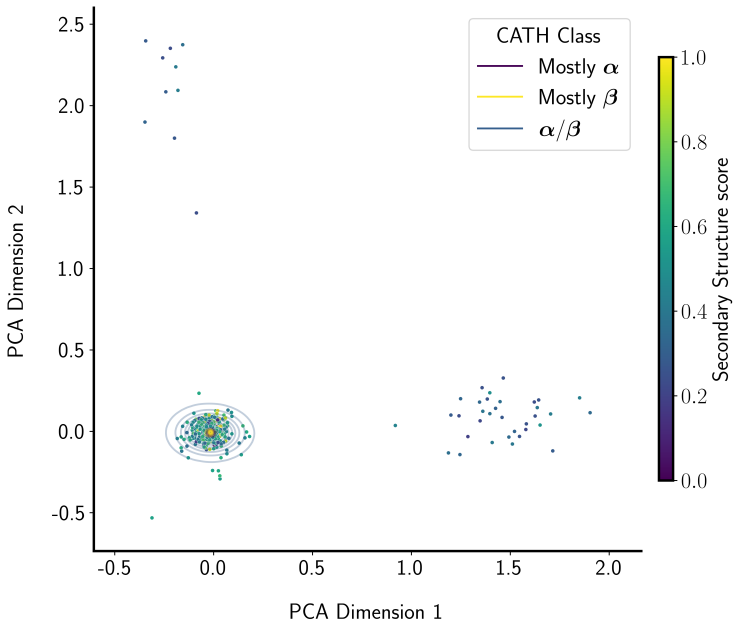
(b) Colored by averaged electro-negativity values



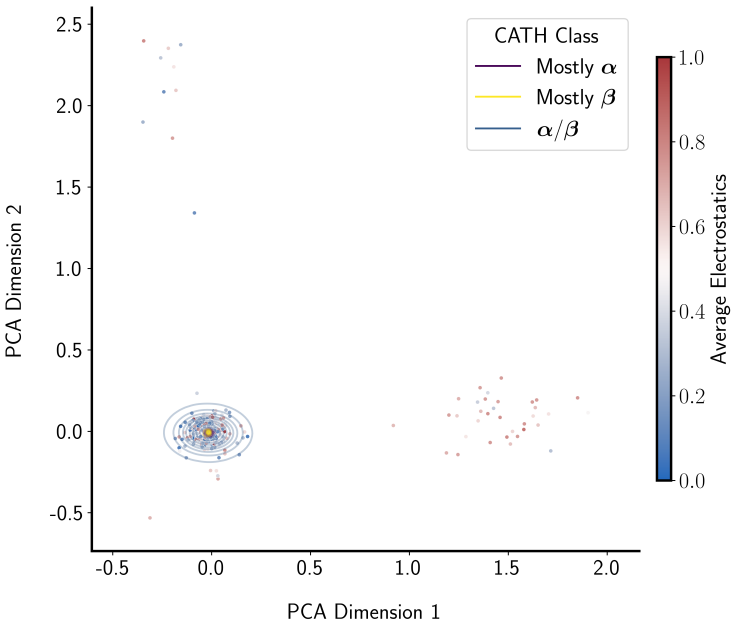
(c) Colored by true CATH Superfamily

Figure 10: Latent Space from t-SNE

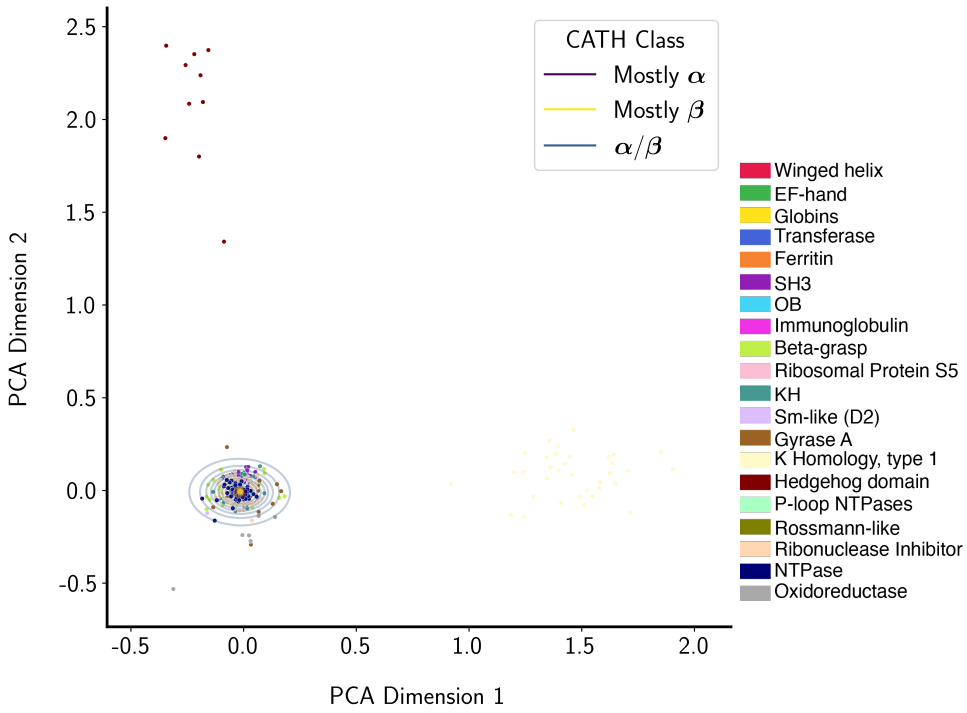
5.3 PCA



(a) Colored by secondary structure



(b) Colored by averaged electro-negativity values



(c) Colored by true CATH Superfamily

Figure 11: Latent Space from PCA

6 Stochastic Block Modelling

# of Clusters	20
Silhouette Score	−0.0705
Davies-Boundin Score	33.1678
Overlap Score	0.2798
Rand Score	0.8542
Rand Score Adjusted	0.1977
Adjusted Mutual Information	0.4108
Homogeneity Score	0.4831
Completeness Score	0.3749

Table 1: Clustering metrics of DeepUrfold SBM vs CATH

6.1 Electrostatics

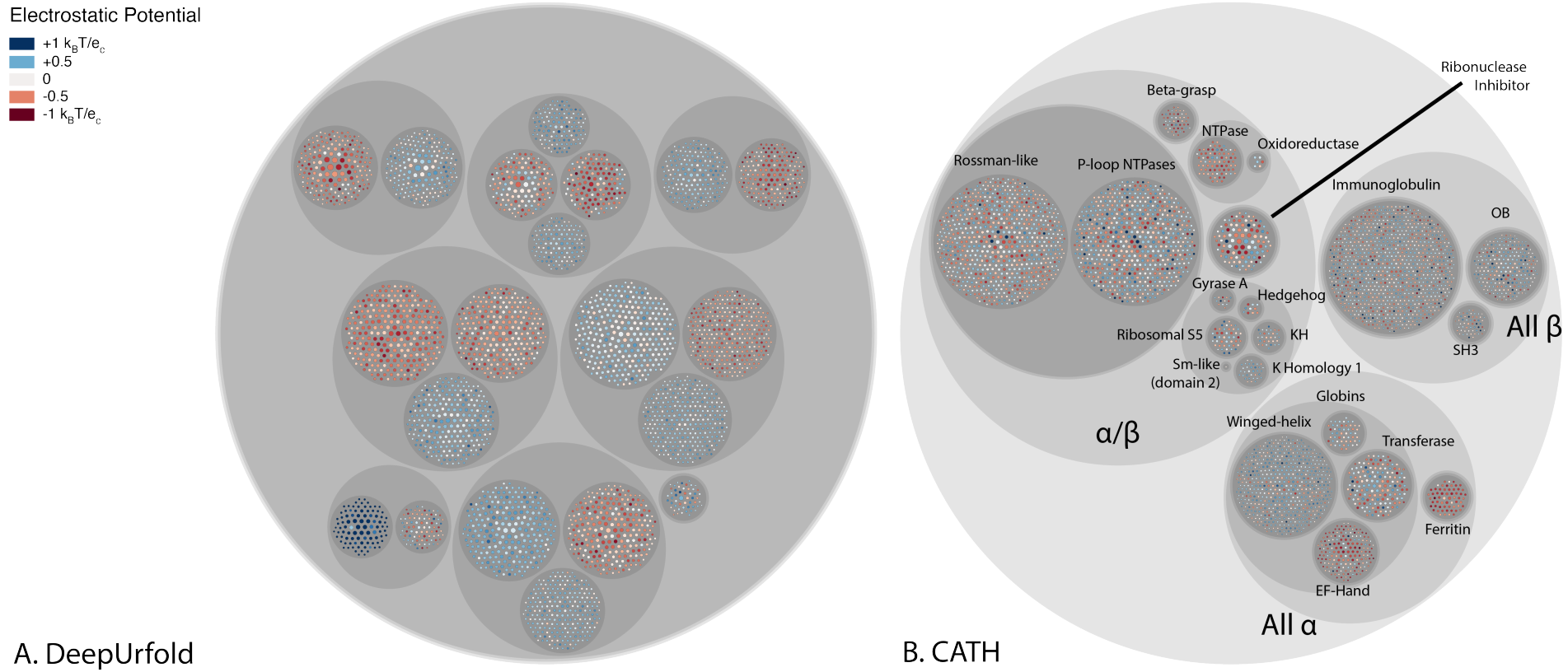


Figure 12: Each atom it annotated with the Boolean feature is_electronegative. We sum up all of the electronegtive atoms and take a fraction of the total number of atoms.

6.2 Charge

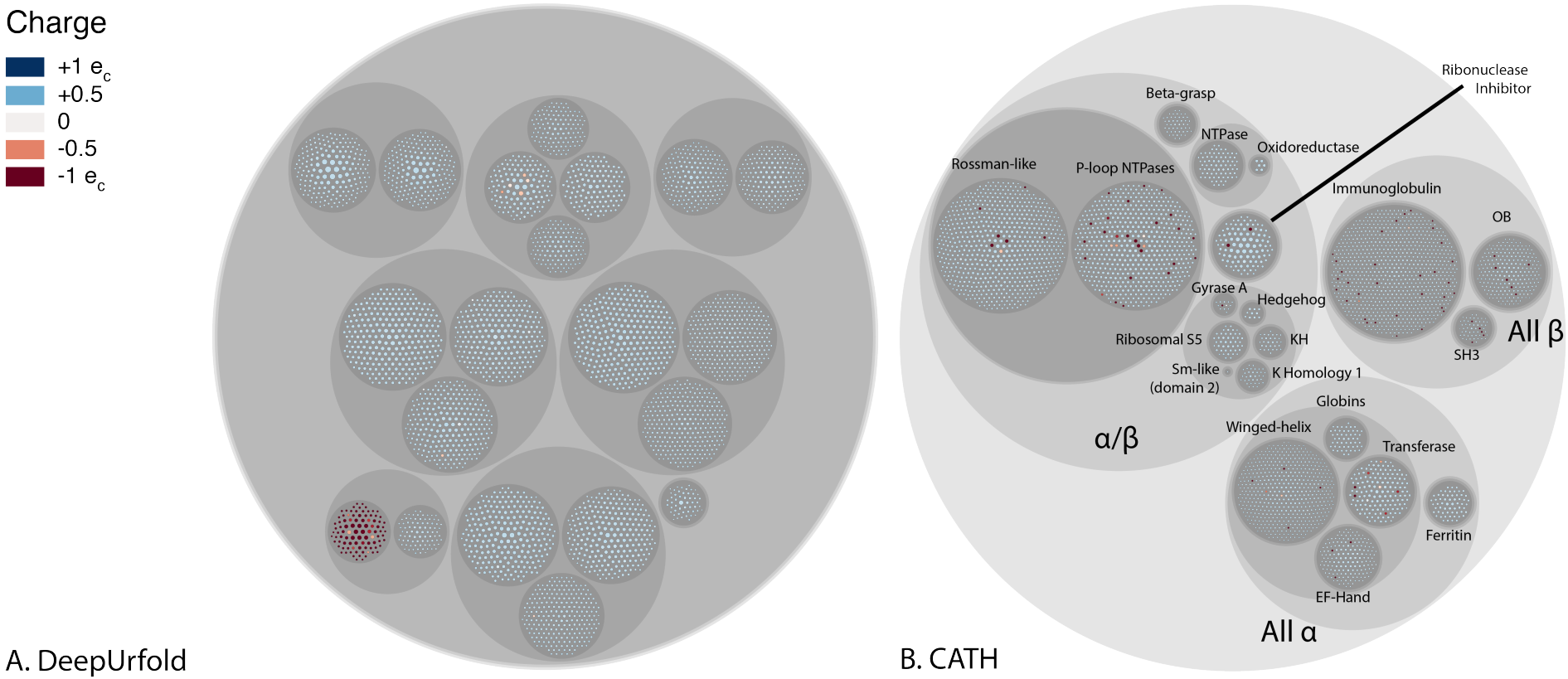


Figure 13: Each atom it annotated with the Boolean feature is_positive. We sum up all of the positive atoms and take a fraction of the total number of atoms.

6.3 Secondary Structure

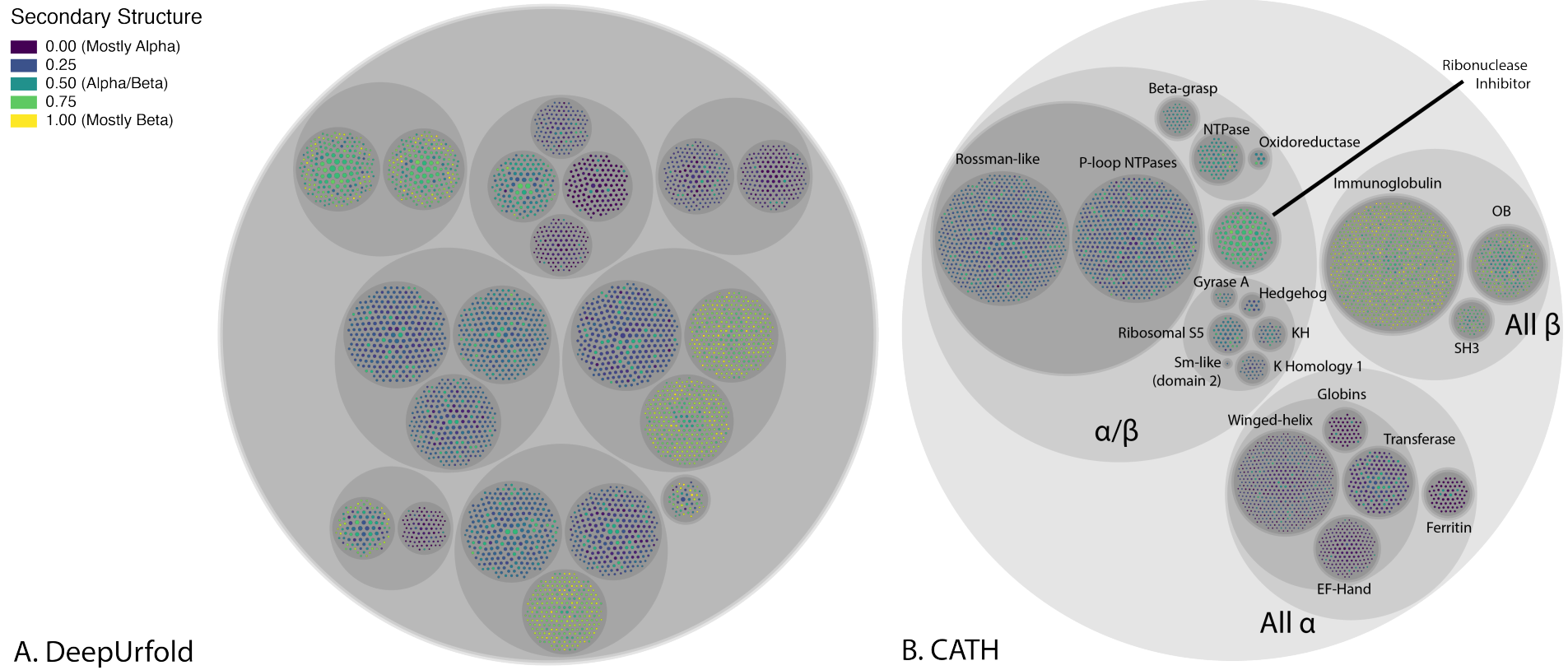


Figure 14: To calculate the secondary structure score, we use the formula: (# beta atoms - # alpha atoms)/(2*(# beta atoms + # alpha atoms)) + 0.5.

6.4 GO: Molecular Function

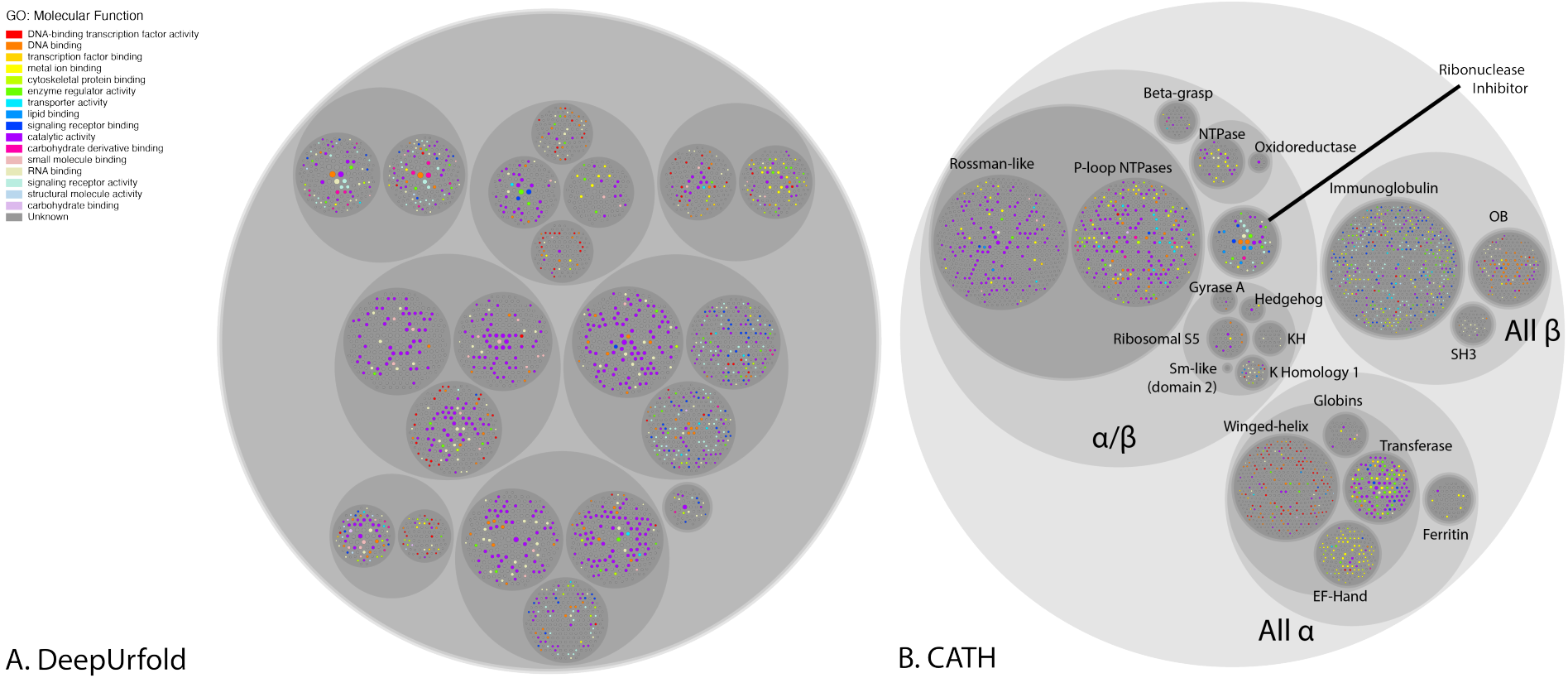


Figure 15: We use GOATOOLS to calculate enrichment for each GO term from all domains in the predicted SBM community (leaf grouping only) using GO Slim terms from AGR. If the domain has a GO term that is enriched in its community (p_fdr_bh ≤ 0.05), then it is colored for the associated term. If there are multiple enriched terms, only the first is used.

6.5 GO: Biological Process

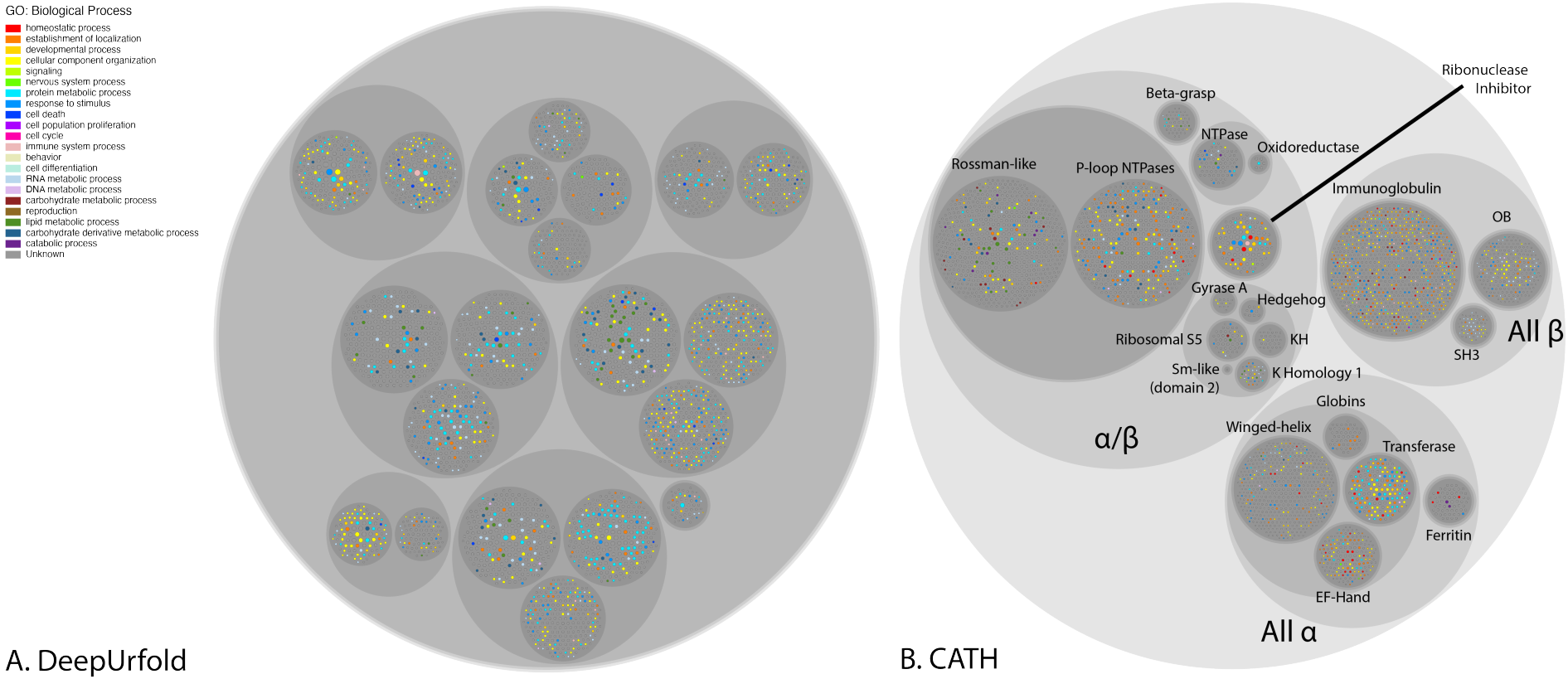


Figure 16: We use GOATOOLS to calculate enrichment for each GO term from all domains in the predicted SBM community (leaf grouping only) using GO Slim terms from AGR. If the domain has a GO term that is enriched in its community (p_fdr_bh ≤ 0.05), then it is colored for the associated term. If there are multiple enriched terms, only the first is used.

6.6 GO: Cellular Component

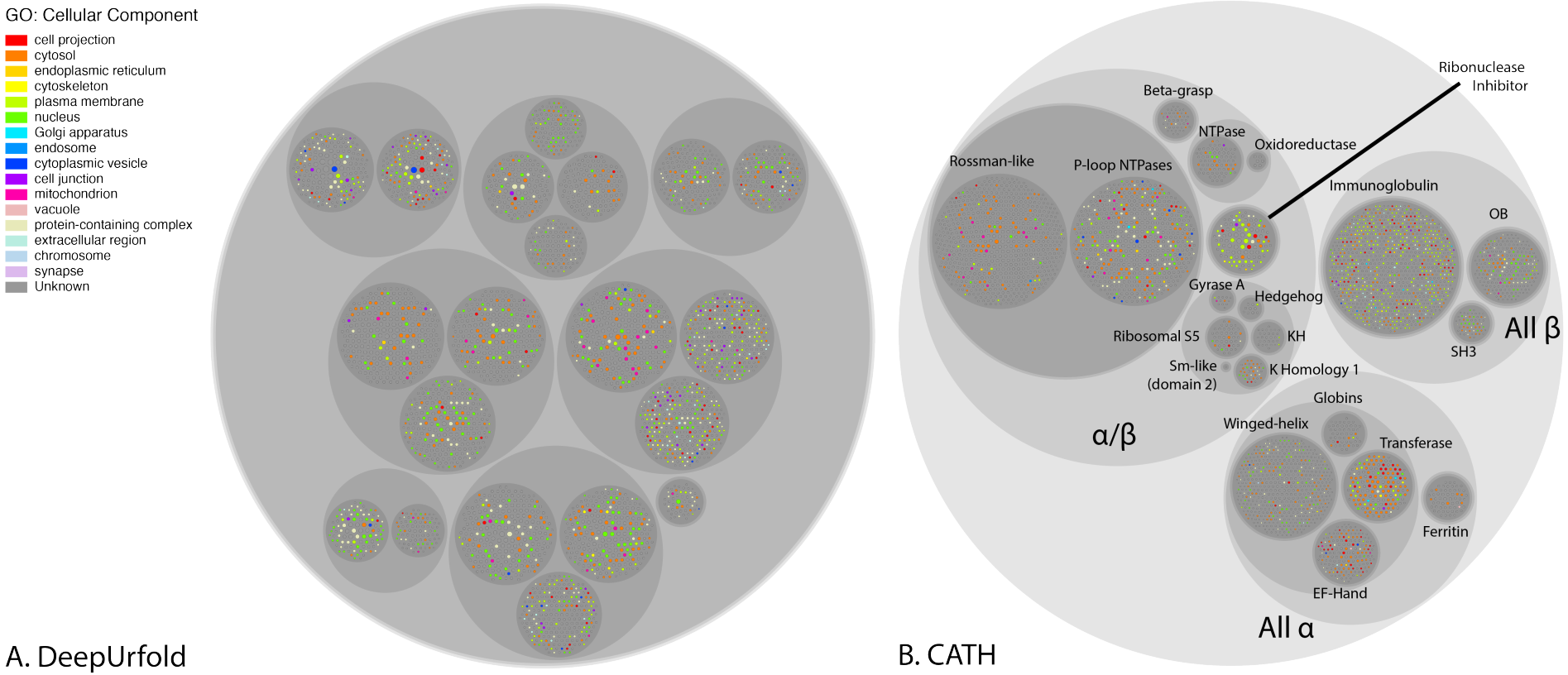


Figure 17: We use GOATOOLS to calculate enrichment for each GO term from all domains in the predicted SBM community (leaf grouping only) using GO Slim terms from AGR. If the domain has a GO term that is enriched in its community (p_fdr_bh ≤ 0.05), then it is colored for the associated term. If there are multiple enriched terms, only the first is used.

6.7 Downsampled SBM

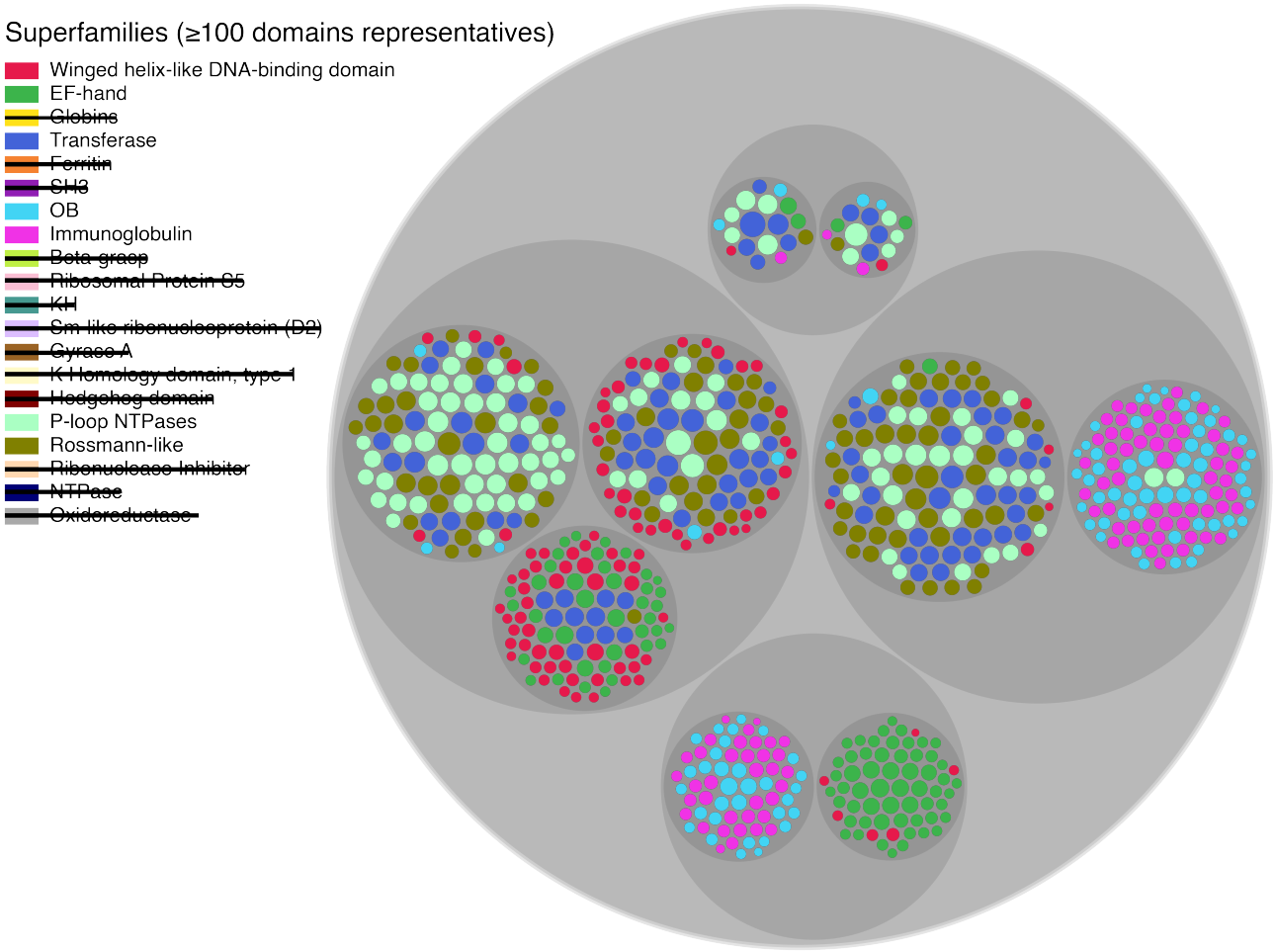


Figure 18: In order to test how the SBM treats highly imbalanced classes, we included only superfamilies that had ≥ 100 domain representatives and sampled 100 random domains from each. No immediate change can be detected and OB domains are still found in the same community as Immunoglobulins

Table 2: Comparison of Stochastic Block Modelling of a Bipartite graph of CATH Domains and Superfamilies with scores based on similar algorithms to DeepUrfold. Silhouette Score and Davis-Boundin are self measures and do not compare against CATH.

	Method	**Model**	**Train Data**	**Score**	**# Communities**	*Silhouette Score*	*Davis-Boundin*	*Overlap*	*Rand Score (Adjusted)*	*Adjusted Mutual Information*	*Homogeneity*	*Completeness*
Sequence-based												
Pairwise	UCLUST	Global Alignment	CATH S35	↑% Seq ID	19	0.2863	2.4830	0.6148	0.9131 (0.6778)	0.7703	0.7712	0.7791
		Local Alignment	CATH S35	↑% Seq ID	38	−0.2584	10.4408	0.3476	0.8338 (0.2730)	0.4382	0.5074	0.4156
Single Model	ESM-1b	Language Transformer	Uniref50	↓Euclidean Distance	121	0.4299	0.7931	0.1676	0.8638 (0.0896)	0.5843	0.9065	0.4311
Superfamily-specific models	HMMER	HMM	CATH S35	↑Bitscore	24	−0.1715	5.3797	0.4122	0.7463 (0.2444)	0.4747	0.4429	0.5416
			EVcouplings alignment	↑Bitscore	14	0.5422	3.5902	0.3105	0.5546 (0.0905)	0.2173	0.1734	0.3382
	SeqDesign	Auto-regressive	EVcouplings alignment (Unaligned)	↑Bitscore	7	0.0410	3.6629	0.4095	0.7604 (0.2085)	0.2736	0.2370	0.3235
Structure-based												
Pairwise	TM-Align	Structural Alignment	CATH S35	↑TM-Score	42	0.1303	1.7138	0.3150	0.8734 (0.2380)	0.6232	0.8247	0.5188
				↓RMSD	29	0.1593	1.8141	0.4200	0.8831 (0.3342)	0.6628	0.8172	0.5706
		Circular Permutations	CATH S35	↑TM-Score	42	0.1124	1.7407	0.2969	0.8702 (0.2162)	0.6042	0.8005	0.5037
				↓RMSD	26	0.1801	1.6529	0.4164	0.8799 (0.3467)	0.6740	0.8032	0.5929
Superfamily-specific models	Struct2Seq	Graph Transformer	CATH (Backbone)	↑Perplexity	15	0.0563	2.5657	0.4724	0.8745 (0.3959)	0.5434	0.5857	0.5196
	DeepUrfold (ours)	3D-CNN VAE	CATH (Full)	↓ELBO	20	−0.0705	33.1678	0.2798	0.8542 (0.1977)	0.4108	0.4831	0.3749

8 Model Architecture

```
DomainStructureVAE(
  (encoder): Encoder(
    (block1): Sequential(
      (0): MinkowskiConvolution(in=20, out=16, kernel_size
          =[3, 3, 3], stride=[2, 2, 2], dilation=[1, 1,
          1])
      (1): MinkowskiSyncBatchNorm(16, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (2): MinkowskiELU()
      (3): MinkowskiConvolution(in=16, out=16, kernel_size
          =[3, 3, 3], stride=[1, 1, 1], dilation=[1, 1,
          1])
      (4): MinkowskiSyncBatchNorm(16, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (5): MinkowskiELU()
    )
    (block2): Sequential(
      (0): MinkowskiConvolution(in=16, out=32, kernel_size
          =[3, 3, 3], stride=[2, 2, 2], dilation=[1, 1,
          1])
      (1): MinkowskiSyncBatchNorm(32, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (2): MinkowskiELU()
      (3): MinkowskiConvolution(in=32, out=32, kernel_size
          =[3, 3, 3], stride=[1, 1, 1], dilation=[1, 1,
          1])
      (4): MinkowskiSyncBatchNorm(32, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (5): MinkowskiELU()
    )
    (block3): Sequential(
      (0): MinkowskiConvolution(in=32, out=64, kernel_size
          =[3, 3, 3], stride=[2, 2, 2], dilation=[1, 1,
          1])
      (1): MinkowskiSyncBatchNorm(64, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (2): MinkowskiELU()
      (3): MinkowskiConvolution(in=64, out=64, kernel_size
          =[3, 3, 3], stride=[1, 1, 1], dilation=[1, 1,
          1])
      (4): MinkowskiSyncBatchNorm(64, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (5): MinkowskiELU()
    )
    (block4): Sequential(
      (0): MinkowskiConvolution(in=64, out=128,
          kernel_size=[3, 3, 3], stride=[2, 2, 2],
          dilation=[1, 1, 1])
      (1): MinkowskiSyncBatchNorm(128, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (2): MinkowskiELU()
      (3): MinkowskiConvolution(in=128, out=128,
          kernel_size=[3, 3, 3], stride=[1, 1, 1],
          dilation=[1, 1, 1])
      (4): MinkowskiSyncBatchNorm(128, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (5): MinkowskiELU()
    )
    (block5): Sequential(
      (0): MinkowskiConvolution(in=128, out=256,
          kernel_size=[3, 3, 3], stride=[2, 2, 2],
          dilation=[1, 1, 1])
      (1): MinkowskiSyncBatchNorm(256, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (2): MinkowskiELU()
      (3): MinkowskiConvolution(in=256, out=256,
          kernel_size=[3, 3, 3], stride=[1, 1, 1],
          dilation=[1, 1, 1])
      (4): MinkowskiSyncBatchNorm(256, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (5): MinkowskiELU()
    )
    (block6): Sequential(
      (0): MinkowskiConvolution(in=256, out=512,
          kernel_size=[3, 3, 3], stride=[2, 2, 2],
          dilation=[1, 1, 1])
      (1): MinkowskiSyncBatchNorm(512, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (2): MinkowskiELU()
      (3): MinkowskiConvolution(in=512, out=512,
          kernel_size=[3, 3, 3], stride=[1, 1, 1],
          dilation=[1, 1, 1])
      (4): MinkowskiSyncBatchNorm(512, eps=1e-05, momentum
          =0.1, affine=True, track_running_stats=True)
      (5): MinkowskiELU()
    )
    (block7): Sequential(
      (0): MinkowskiConvolution(in=512, out=1024,
          kernel_size=[3, 3, 3], stride=[2, 2, 2],
          dilation=[1, 1, 1])
      (1): MinkowskiSyncBatchNorm(1024, eps=1e-05,
          momentum=0.1, affine=True, track_running_stats=
          True)
      (2): MinkowskiELU()
      (3): MinkowskiConvolution(in=1024, out=1024,
          kernel_size=[3, 3, 3], stride=[1, 1, 1],
          dilation=[1, 1, 1])
      (4): MinkowskiSyncBatchNorm(1024, eps=1e-05,
          momentum=0.1, affine=True, track_running_stats=
          True)
      (5): MinkowskiELU()
    )
    (global_pool): MinkowskiGlobalPooling(mode=PoolingMode
        .GLOBAL_AVG_POOLING_PYTORCH_INDEX)
    (linear_mean): MinkowskiLinear(in_features=1024,
        out_features=1024, bias=True)
    (linear_log_var): MinkowskiLinear(in_features=1024,
        out_features=1024, bias=True)
  )
  (decoder): Decoder(
    (block1): Sequential(
      (0): MinkowskiConvolutionTranspose(in=1024, out
          =1024, kernel_size=[2, 2, 2], stride=[2, 2, 2],
          dilation=[1, 1, 1])
      (1): MinkowskiSyncBatchNorm(1024, eps=1e-05,
          momentum=0.1, affine=True, track_running_stats=
          True)
      (2): MinkowskiELU()
      (3): MinkowskiConvolution(in=1024, out=1024,
          kernel_size=[3, 3, 3], stride=[1, 1, 1],
          dilation=[1, 1, 1])
      (4): MinkowskiSyncBatchNorm(1024, eps=1e-05,
          momentum=0.1, affine=True, track_running_stats=
          True)
      (5): MinkowskiELU()
      (6): MinkowskiConvolutionTranspose(in=1024, out=512,
```

```
      kernel_size=[2, 2, 2], stride=[2, 2, 2],
      dilation=[1, 1, 1])
  (7): MinkowskiSyncBatchNorm(512, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (8): MinkowskiELU()
  (9): MinkowskiConvolution(in=512, out=512,
      kernel_size=[3, 3, 3], stride=[1, 1, 1],
      dilation=[1, 1, 1])
  (10): MinkowskiSyncBatchNorm(512, eps=1e-05,
      momentum=0.1, affine=True, track_running_stats=
      True)
  (11): MinkowskiELU()
)
(block2): Sequential(
  (0): MinkowskiConvolutionTranspose(in=512, out=256,
      kernel_size=[2, 2, 2], stride=[2, 2, 2],
      dilation=[1, 1, 1])
  (1): MinkowskiSyncBatchNorm(256, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (2): MinkowskiELU()
  (3): MinkowskiConvolution(in=256, out=256,
      kernel_size=[3, 3, 3], stride=[1, 1, 1],
      dilation=[1, 1, 1])
  (4): MinkowskiSyncBatchNorm(256, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (5): MinkowskiELU()
)
(block3): Sequential(
  (0): MinkowskiConvolutionTranspose(in=256, out=128,
      kernel_size=[2, 2, 2], stride=[2, 2, 2],
      dilation=[1, 1, 1])
  (1): MinkowskiSyncBatchNorm(128, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (2): MinkowskiELU()
  (3): MinkowskiConvolution(in=128, out=128,
      kernel_size=[3, 3, 3], stride=[1, 1, 1],
      dilation=[1, 1, 1])
  (4): MinkowskiSyncBatchNorm(128, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (5): MinkowskiELU()
)
(block4): Sequential(
  (0): MinkowskiConvolutionTranspose(in=128, out=64,
      kernel_size=[2, 2, 2], stride=[2, 2, 2],
      dilation=[1, 1, 1])
  (1): MinkowskiSyncBatchNorm(64, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (2): MinkowskiELU()
  (3): MinkowskiConvolution(in=64, out=64, kernel_size
      =[3, 3, 3], stride=[1, 1, 1], dilation=[1, 1,
      1])
  (4): MinkowskiSyncBatchNorm(64, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (5): MinkowskiELU()
)
(block5): Sequential(
  (0): MinkowskiConvolutionTranspose(in=64, out=32,
      kernel_size=[2, 2, 2], stride=[2, 2, 2],
      dilation=[1, 1, 1])
  (1): MinkowskiSyncBatchNorm(32, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (2): MinkowskiELU()
  (3): MinkowskiConvolution(in=32, out=32, kernel_size
      =[3, 3, 3], stride=[1, 1, 1], dilation=[1, 1,
      1])
  (4): MinkowskiSyncBatchNorm(32, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (5): MinkowskiELU()
)
(block6): Sequential(
  (0): MinkowskiConvolutionTranspose(in=32, out=16,
      kernel_size=[2, 2, 2], stride=[2, 2, 2],
      dilation=[1, 1, 1])
  (1): MinkowskiSyncBatchNorm(16, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (2): MinkowskiELU()
  (3): MinkowskiConvolution(in=16, out=16, kernel_size
      =[3, 3, 3], stride=[1, 1, 1], dilation=[1, 1,
      1])
  (4): MinkowskiSyncBatchNorm(16, eps=1e-05, momentum
      =0.1, affine=True, track_running_stats=True)
  (5): MinkowskiELU()
)
(block7): MinkowskiConvolution(in=16, out=20,
    kernel_size=[1, 1, 1], stride=[1, 1, 1], dilation
    =[1, 1, 1])
(pruning): MinkowskiPruning()
)
```