Supplement

A. Input data statistics

We use Swiss-Prot version 2019_01 in our analysis, which gives us 559077 proteins, or 548264 after filtering for only 20 standard amino acids and filtering fragments.

Because different protein functions have differing prevalence, we note the number of proteins that have a given function for Pfam, EC, and GO labels, as well as noting the number of labels per protein.

Fold	Number of sequences
train	438522
dev	55453
test	54289
all together	548264

Table S1. In our random split of the training data, we allocate about 80% to the training fold, 10% to the development fold, and 10% to the test fold.

Fold	Number of sequences
train	182965
dev	180309
test	183475
all together	546749

Table S2. In our clustered split of the training data, we use UniRef50, and allocate approximately equal numbers of sequences to each fold.

~~Because different protein functions have differing prevalence, we note the number of proteins that have a given function for Pfam, EC, and GO labels, as well as noting the number of labels per protein.~~

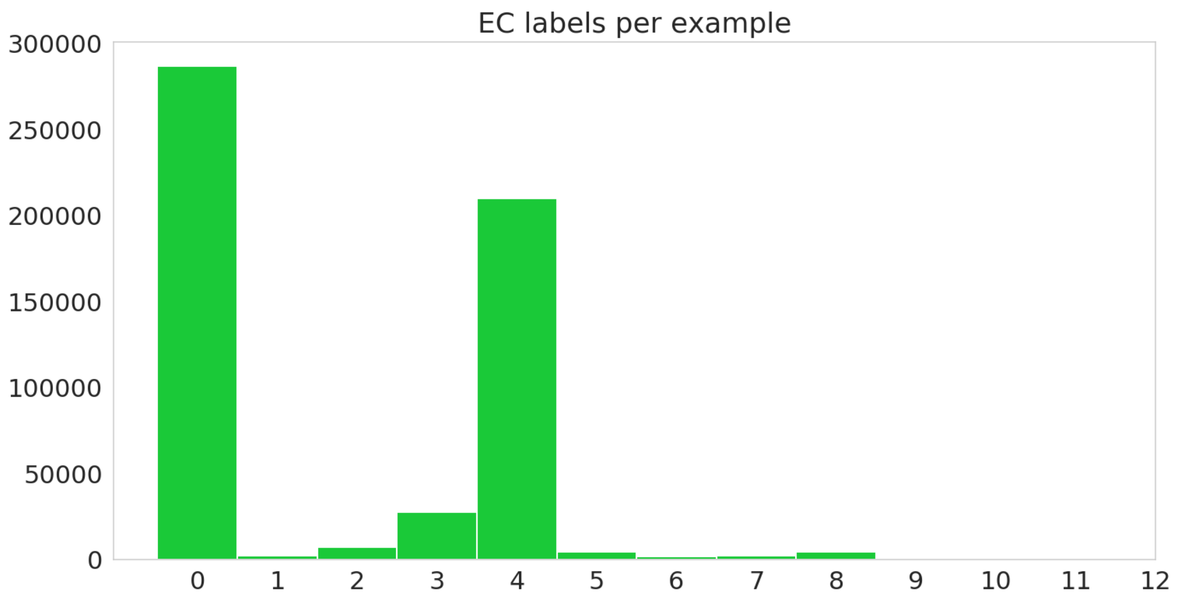


Fig. S1. Histogram of number of labels per sequence, including hierarchical labels, on the random dataset.

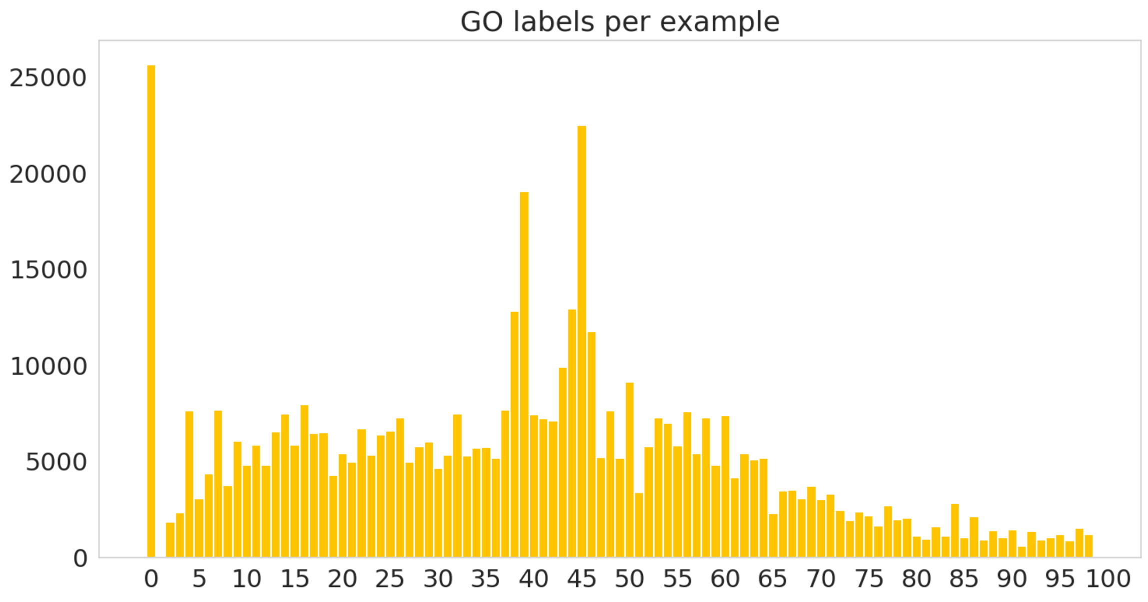


Fig. S2. Histogram of number of labels per sequence, including hierarchical labels, on the random dataset.

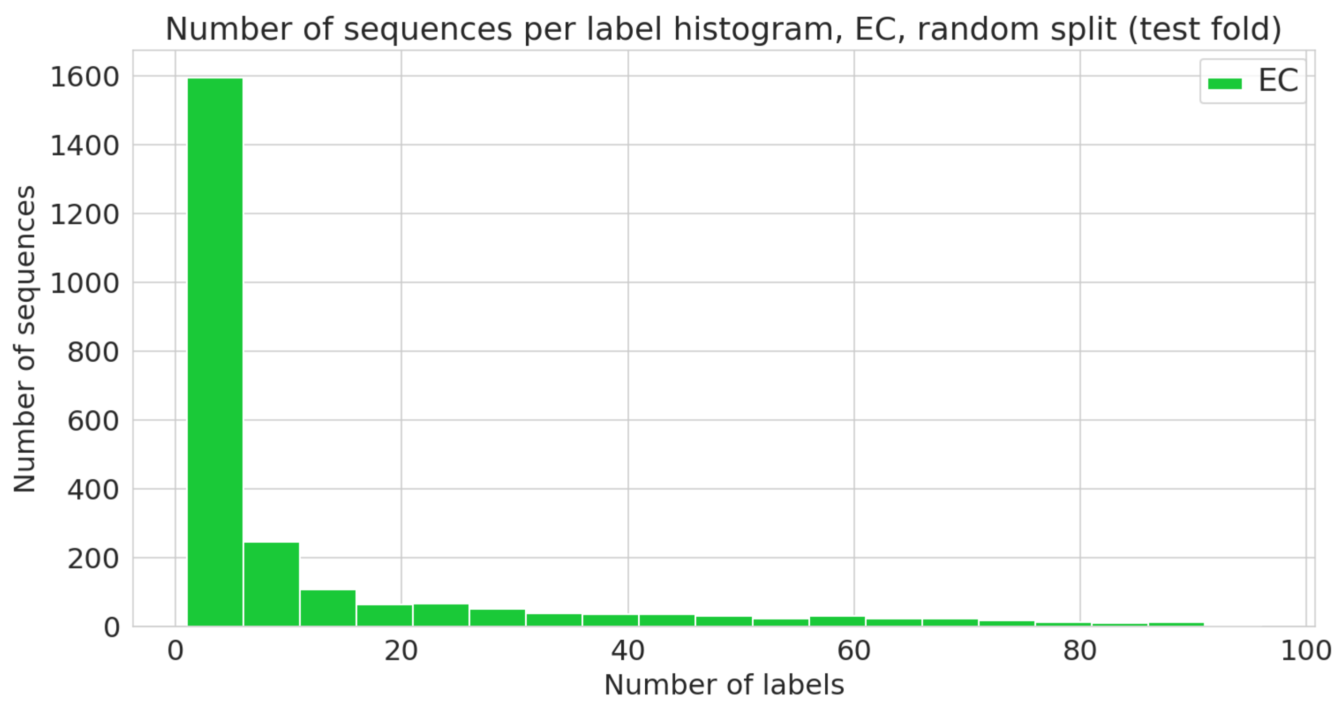


Fig. S3. Number of sequences annotated with a given functional label. (EC class) in the random dataset.

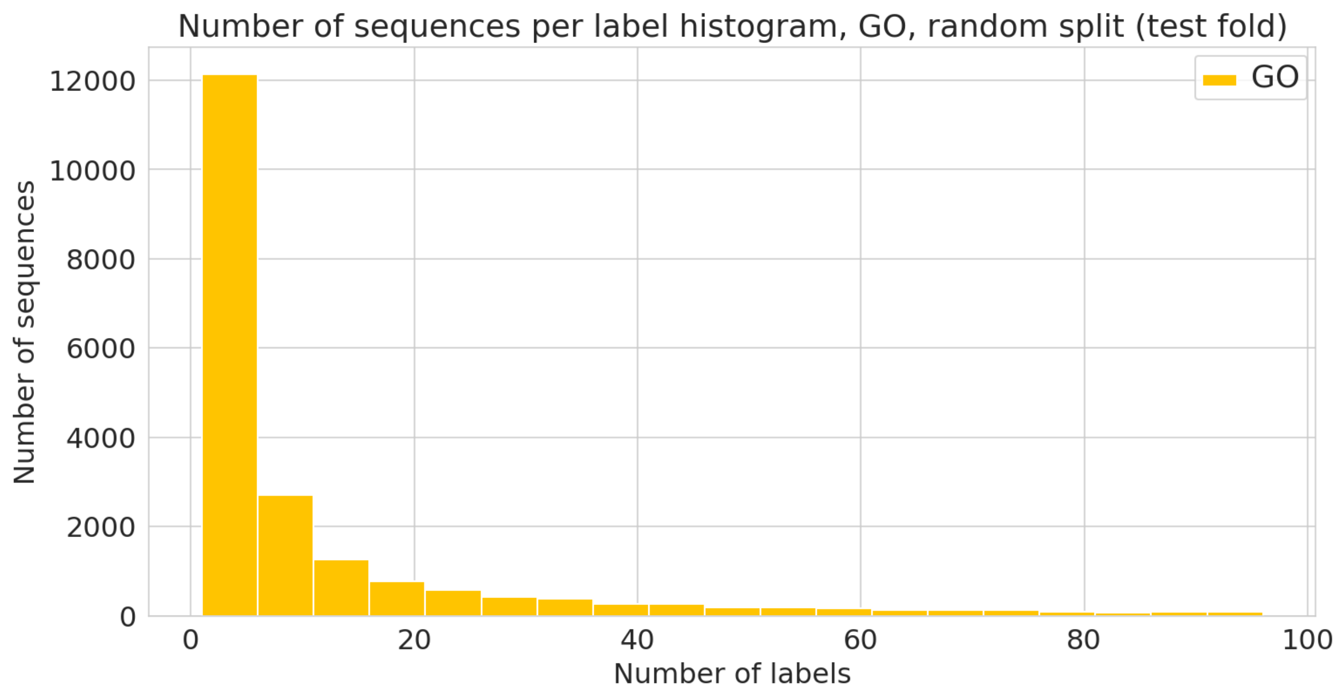


Fig. S4. Number of sequences annotated with a given functional label. (GO label) in the random dataset.

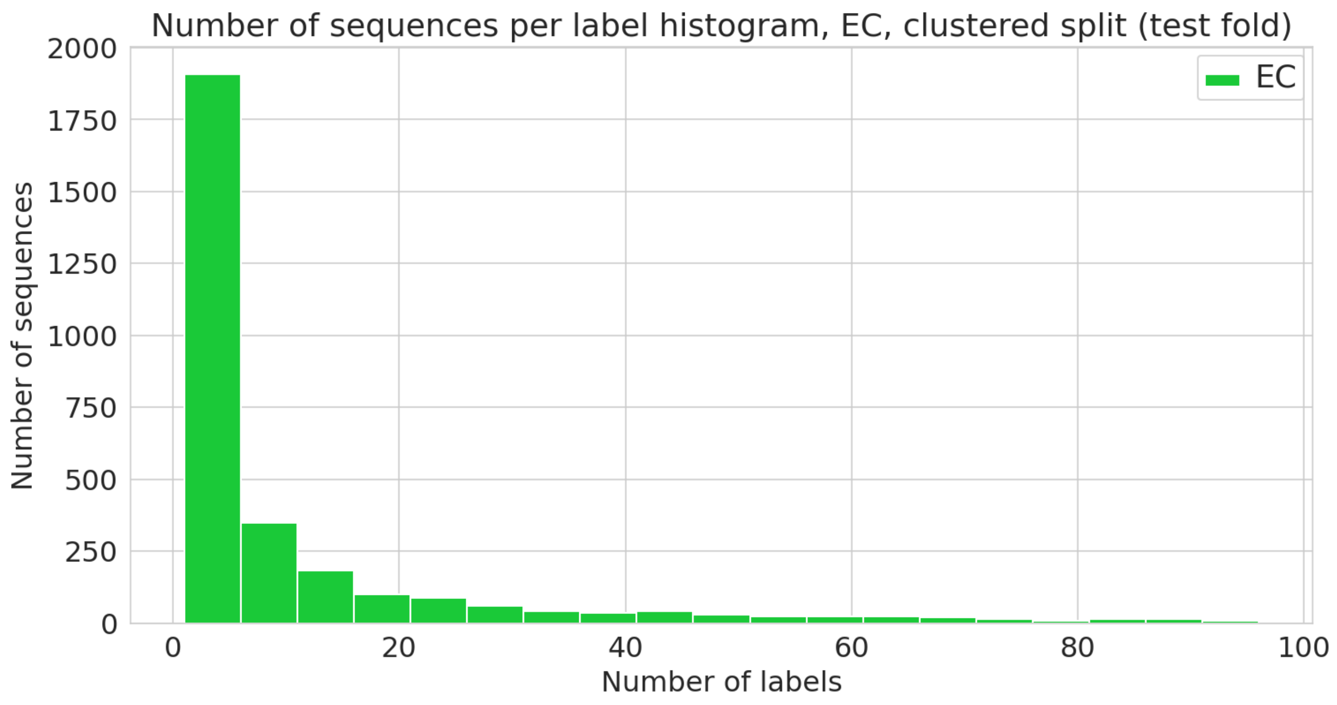


Fig. S5. Number of sequences annotated with a given functional label. (EC class) in the clustered dataset.

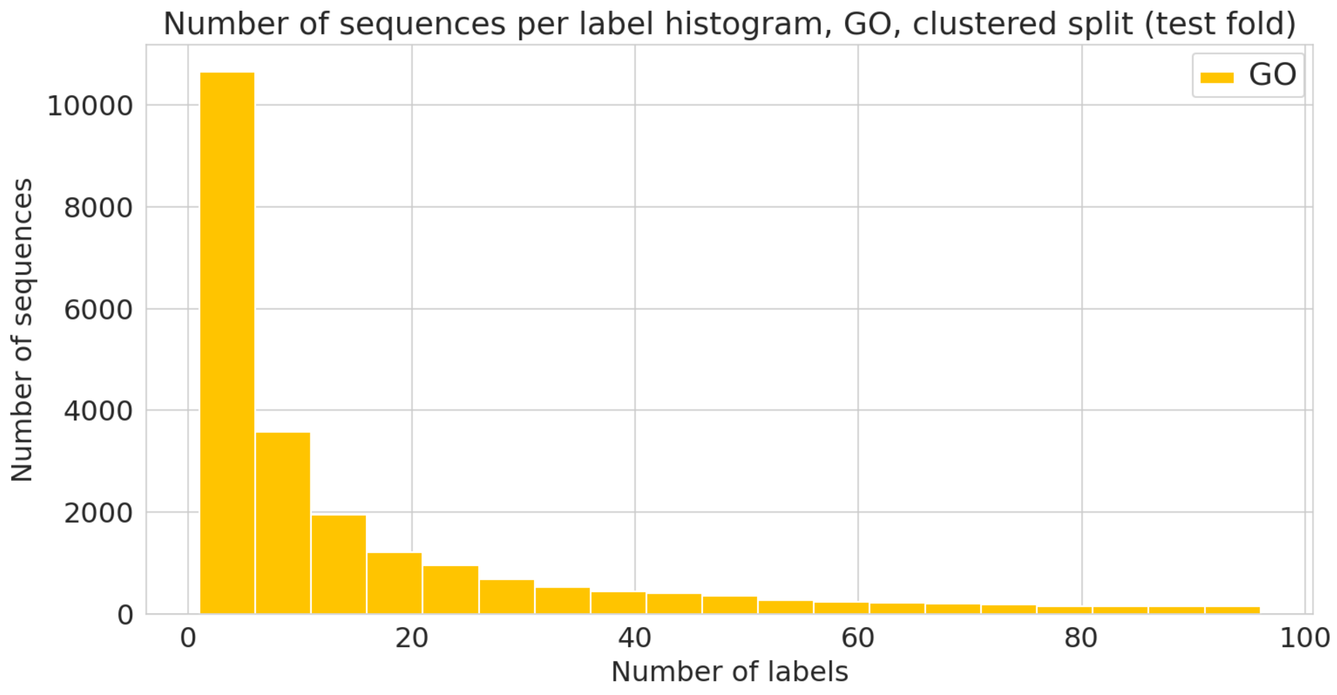


Fig. S6. Number of sequences annotated with a given functional label. (GO label) in the clustered dataset.

When assigning examples to folds in our clustered dataset, we note that there are test examples that have labels that are never seen in the training data. We report these cases below as “Impossible” test example-label pairs.

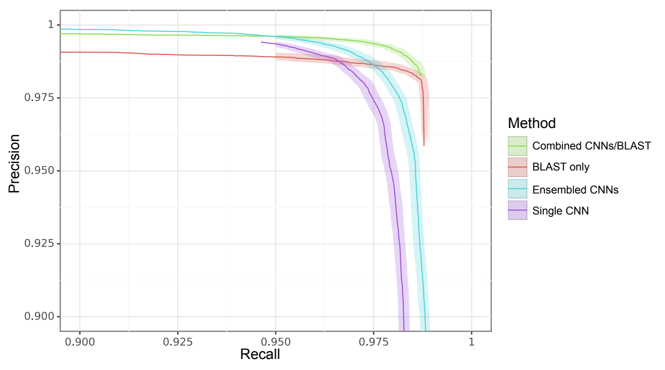
Type	Number
Train labels	3411
Test labels	3414
Impossible test labels	1043
Train example-label pairs	348105
Test example-label pairs	348755
Impossible test example-label pairs	3415

Table S3. Clustered dataset statistics for EC labels.

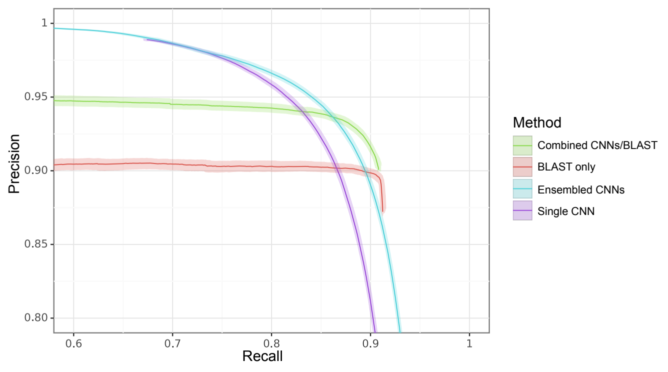
Type	Number
Train labels	26538
Test labels	26666
Impossible test labels	3739
Train example-label pairs	8338584
Test example-label pairs	8424299
Impossible test example-label pairs	11137

Table S4. Clustered dataset statistics for GO labels.

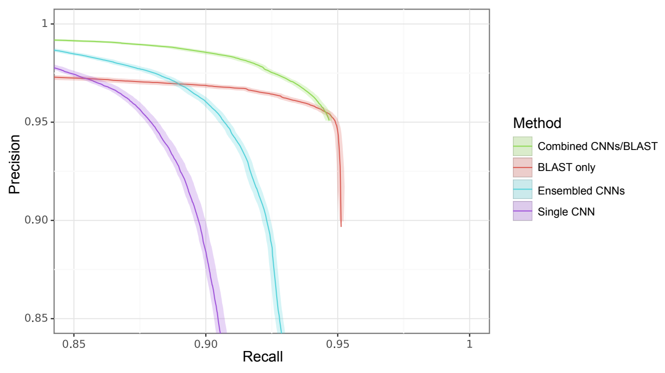
B. Precision/Recall curves



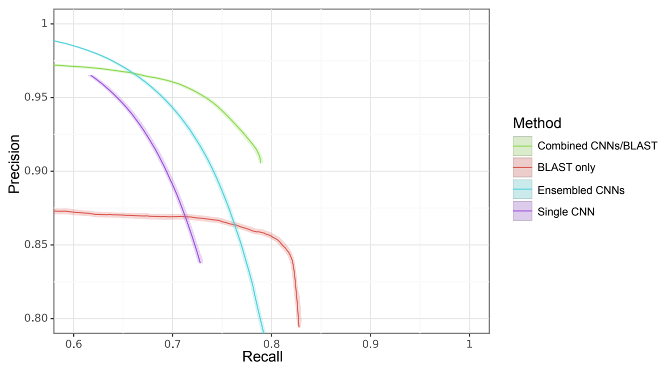
(a) EC number prediction: random split



(b) GO term prediction: random split

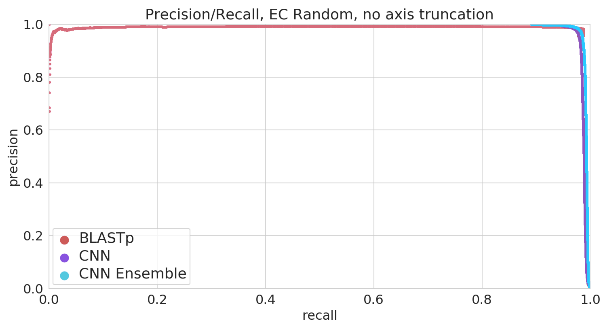


(c) EC number prediction: clustered split

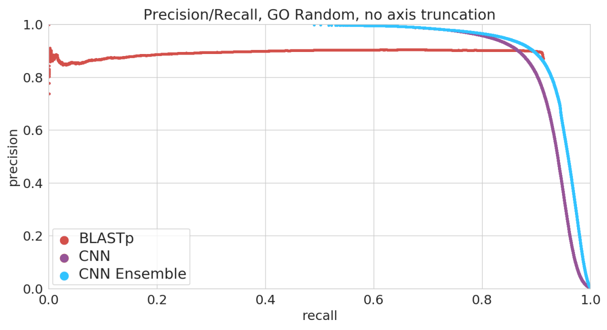


(d) GO term prediction: clustered split

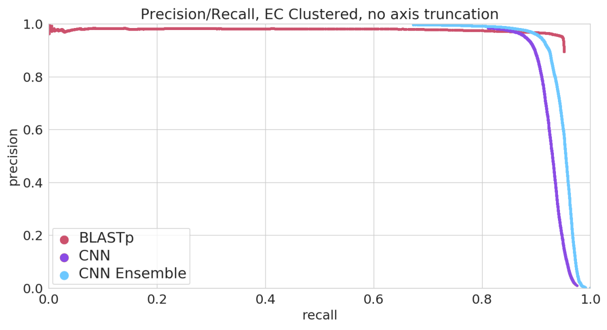
Fig. S7. Bootstrapped precision-recall curves for EC number prediction and gene ontology term prediction for random and clustered splits for four methods: BLAST top pick, single ProteInfer CNN, ensembled ProteInfer CNNs, and ensembled ProteInfer CNNs scaled by BLAST score.



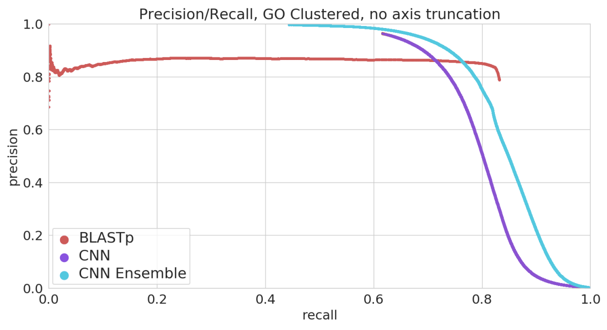
(a) EC number prediction: random split



(b) GO term prediction: random split



(c) EC number prediction: clustered split



(d) GO term prediction: clustered split

Fig. S8. Full precision-recall curves for EC number prediction and gene ontology term prediction for random and clustered splits for four methods: BLAST top pick, single ProteInfer CNN, ensembled ProteInfer CNNs

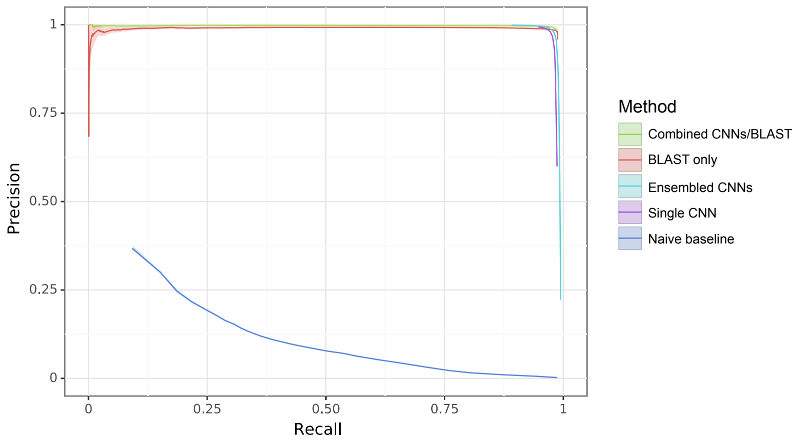
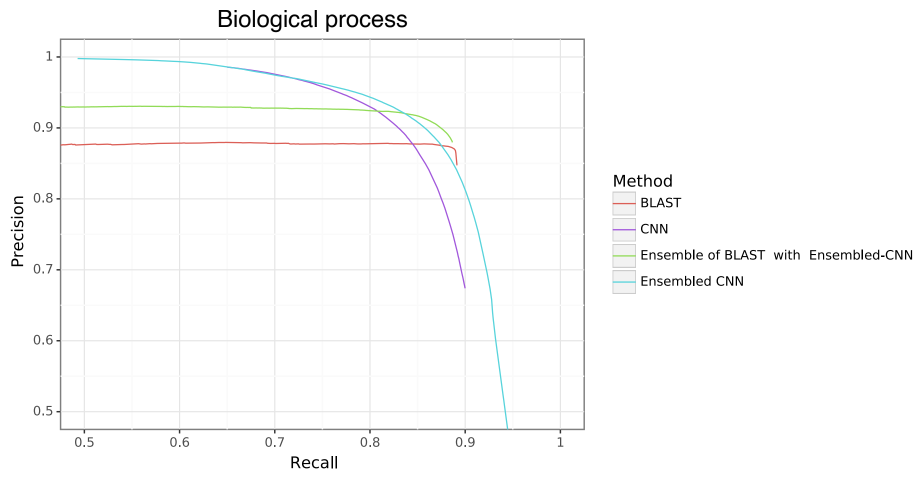
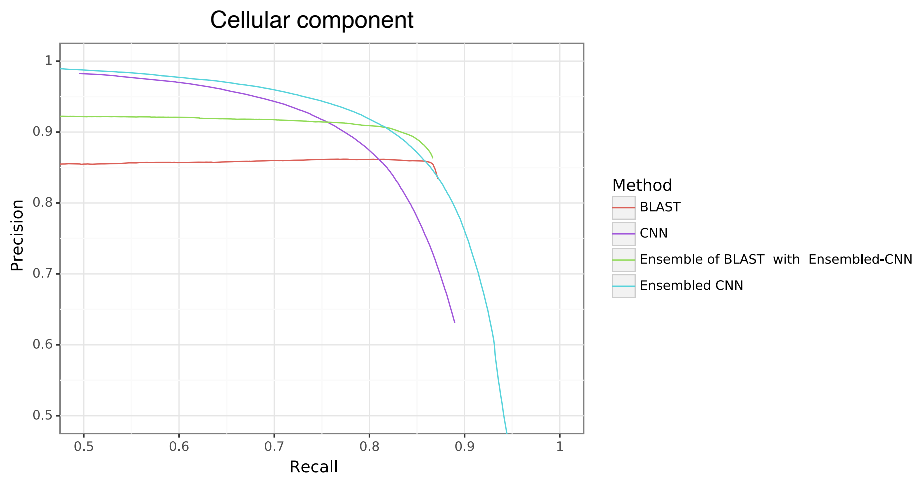


Fig. S9. EC random task with different methods compared against a naive baseline where the predictor is simply the frequency in the training set.

Sanderson *et al.* | ProteInfer bioRχiv | 19

C. GO performance by ontology





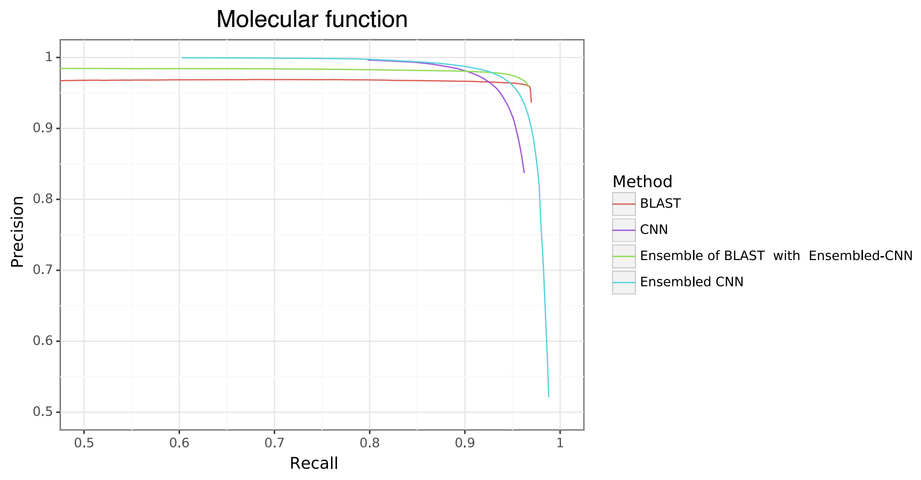


Fig. S10. GO performance stratified by method and ontology type.

D. Stratified performance

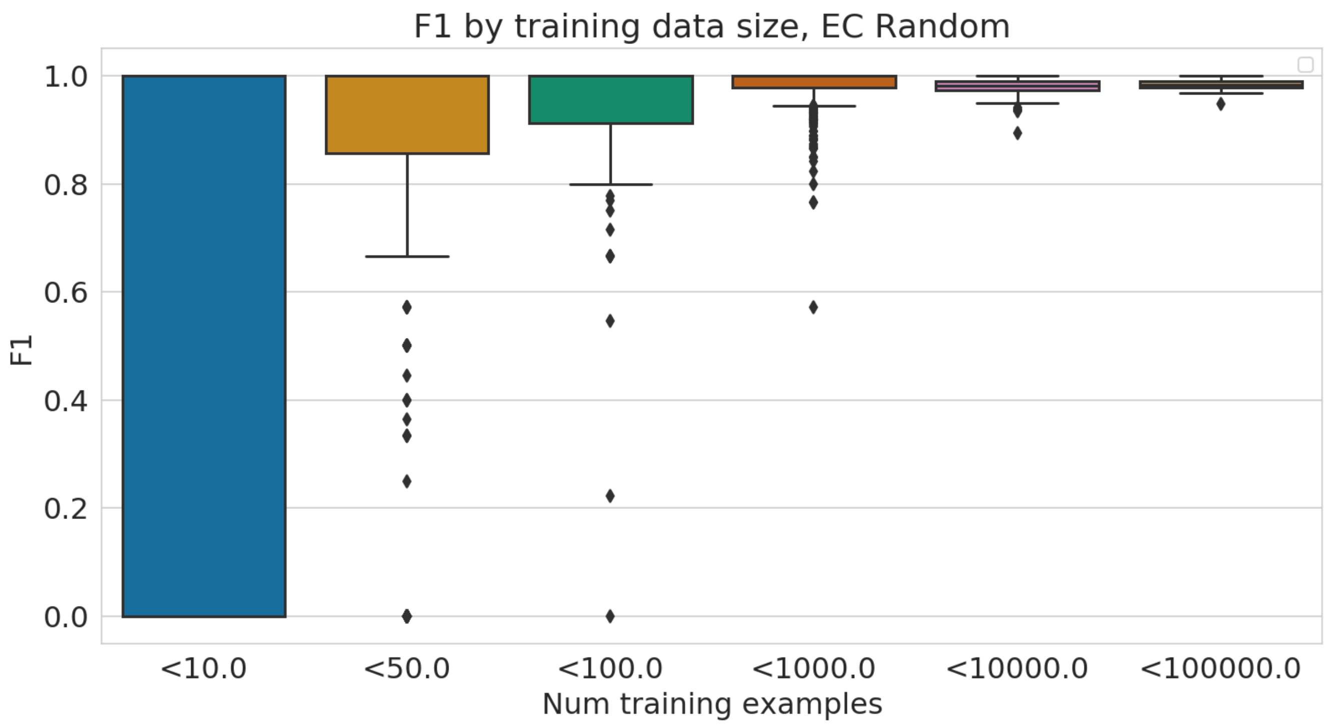


Fig. S11. Performance of EC model stratified by number of training examples available for each test example.

Domain architecture diversity in bifunctional enzymes

First domain	Second Domain	Number ordered correctly	Number times seen	Percent Correct
EC:2.7.7.60	EC:4.6.1.12	94	94	100
EC:4.1.99.12	EC:3.5.4.25	83	83	100
EC:3.5.4.19	EC:3.6.1.31	59	59	100
EC:1.8.4.11	EC:1.8.4.12	20	20	100
EC:4.1.1.48	EC:5.3.1.24	18	18	100
EC:5.4.99.5	EC:4.2.1.51	12	12	100
EC:5.4.99.5	EC:1.3.1.12	4	4	100
EC:4.2.1.10	EC:1.1.1.25	3	3	100
EC:2.7.7.61	EC:2.4.2.52	0	3	0
EC:2.7.1.71	EC:4.2.3.4	0	2	0
EC:1.1.1.25	EC:4.2.1.10	0	1	0
EC:2.7.2.3	EC:5.3.1.1	1	1	100
EC:4.1.1.97	EC:1.7.3.3	1	1	100
EC:4.1.3.1	EC:2.3.3.9	1	1	100
EC:5.1.99.6	EC:1.4.3.5	0	1	0
EC:1.8.4.12	EC:1.8.4.11	0	1	0

Table S5. Domain architecture diversity in bifunctional enzymes. In Swiss-Prot, there are 16 candidate domain architectures available for our EC functional localization experiment. Among these all domain architectures with more than 3 instances in Swiss-Prot (7 of them) are 100% correctly ordered by our CAM method.

E. Vocabulary sizes

Vocabulary	Number of terms
EC	5,134
GO	32,109

Table S6. Vocabulary sizes in models trained for EC and GO.

F. Combining CNN and BLAST

We noticed a clear difference between the performance of CNN models and BLAST models. The CNN models were able make predictions with higher precision than BLAST for a given recall value, but achieved lower overall recall, while BLAST gave high recall at the expense of precision. This suggested that combining the two methods would be beneficial. A CNN, or an ensemble of CNNs produces a metric that is notionally a probability (though often imperfectly calibrated), while BLAST bit-score produces a bit-score metric indicating the significance of the match. We reasoned that one approach to combining the two would simply be to multiply the values together - improving the precision of BLAST predictions by reducing bit-scores in cases where the CNN model lacked confidence in a prediction. This approach performed well, and was the best that we evaluated. To implement this method, all labels associated with the top hit from BLAST with a sequence are initially assigned identical scores, determined by the bit-score of the top match, but these are then rescaled according to the output of the ProteInfer command-line interface.

Whether this approach is worth the additional infrastructure required to maintain two different methods of prediction will depend on the scale of analysis being conducted. Further sophistication in how the CNN and BLAST are combined, perhaps with a learnt model, might further improve performance.

G. Pfam implementation

The ProteInfer algorithm is set up to allow any desired training vocabulary to be used. We demonstrated this by additionally training a model for predicting Pfam families from full-length protein sequences, which is available through our CLI-tool, and performs as shown below.

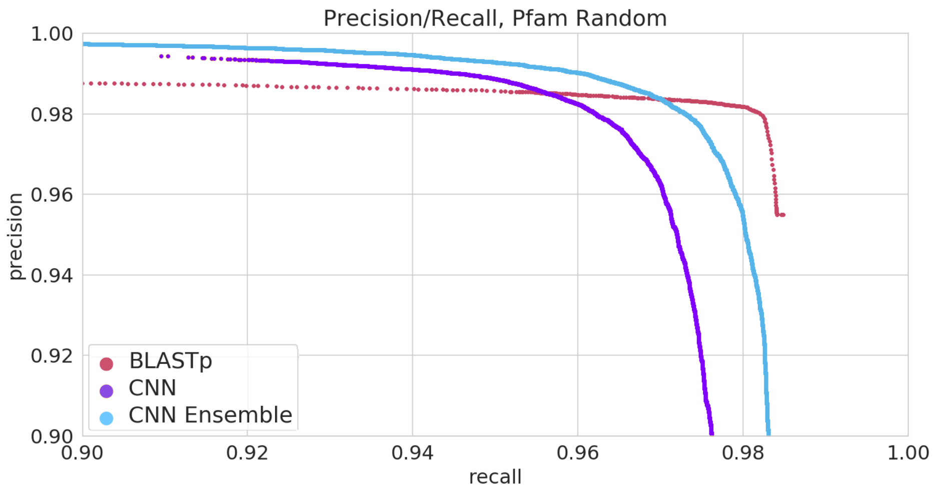


Fig. S12. Performance for Pfam prediction for random train/test split.