nature portfolio

https://doi.org/10.1038/s41587-021-01179-w

Supplementary information

Using deep learning to annotate the protein universe

Using Deep Learning to Annotate the Protein Universe

Maxwell L. Bileschi[1, *], David Belanger[1], Drew H. Bryant[1], Theo Sanderson[1, 2], Brandon Carter[3], D. Sculley[1], Alex Bateman[4], Mark A. DePristo[1, 5], and Lucy J. Colwell[1, 6, *]

[1]Google Research, Cambridge, MA, USA
[2]The Francis Crick Institute, London UK
[3]MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA
[4]European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Hinxton, UK
[5]BigHat Biosciences, San Mateo, CA, USA
[6]Department. of Chemistry, University of Cambridge, Cambridge, UK
[*]Correspondence to mlbileschi@google.com and lcolwell@google.com

Supplementary Information

Pfam data

The positive results obtained using the Top Pick strategy with HMMER, i.e. in the absence of the rigorous statistical filters implemented in HMMER 3.1b2, likely reflect the fact that we are working with sequences that were originally classified by Pfam, and so passed the rigorous statistical thresholds for inclusion. Those sequences that did not pass these filters, and hence were not included in any Pfam family, may pose a more significant challenge to our implementation. For this reason we do not recommend that this HMM implementation is used in settings other than working with these benchmark datasets. For Pfam-full, we do not use the HMMs as a baseline because these models were used to label the data, so may achieve 100% accuracy by default. The Pfam-full dataset has 17772 families overall, and our test and dev sets contain sequences from 16755 families.

Pfam-seed sequences vary between 4 and 2037 amino acids in length, with 27045 seed sequences of length > 500. Fig. S1 contains histograms of Pfam-seed family sizes, the Pfam-seed sequence length distribution and also the frequency of amino acid usage in the Pfam-seed dataset. Family-specific gathering thresholds, shown in Fig. S1C, are used by HMMER 3.1b2 to determine whether a sequence belongs to each family [1]. The role of gathering thresholds is to increase coverage and decrease false positives. However our TPHMM setup simply takes the top match by score, regardless of the assigned gathering threshold. This raises the question of whether implementing the family-specific gathering thresholds would have improved the accuracy score achieved by TPHMM.

To address this, Fig. S1D shows the distribution of top scores for HMMER, for both correct and incorrect predictions. We note that the majority of incorrect predictions have scores *below* the assigned gathering thresholds. However, there also appear to be at least as many correct predictions below these values. This qualitative analysis is backed up by experimentation wherein we determined 8.5% of top picks were below their family-specific gathering thresholds from Pfam. As such, using the assigned gathering thresholds would not have helped performance, and effective use of gathering thresholds would have required us to re-tune each of these values for the training dataset used in this benchmark study.

Pfam allows a domain to belong to multiple families if these families are in the same clan [1]. Our top pick formulation of HMMER for sequence classification does not allow for multiple membership. However, within the seed sequences, there are only two sequences that belong to more than one family. The first sequence has the two distinct names ABEC3_MOUSE/245-418 and E9QMH1_MOUSE/234-407, and the second has the two distinct names NLP_DROME/6-104 and B4HZJ8_DROSE/6-104. Both of these sequences are found in our training dataset.

Definition of sequence identity

Because all sequences within a seed alignment are evolutionarily related it might be that training and test sequences in the random split are very close and thus trivial for the model to accurately classify. To address this, we stratify our analysis by the maximum percent identity of each test sequence with sequences in the train set. Inspired by the method of [2], we use the Pfam-seed family alignments to compute the similarity, measured as percent sequence identity, between every held-out test sequence and the training sequences within the same family. For each pair containing a single test and train sequence we do not realign, but instead use their length L alignment from the seed family multiple sequence alignment. Following [2], for two sequences of n_1 and n_2 residues, if the L aligned sequence position pairs consist in c_{ident} matches, c_{ident} mismatches and c_{ident} cases where either or both sequence contain a gap character such that $L = c_{\text{ident}} + c_{\text{mismat}} + c_{\text{indel}}$, then the pairwise sequence identity is defined as:

$$\text{pid} = \frac{c_{\text{ident}}}{\text{MIN}(n_1, n_2)}.$$

For Pfam-seed we use the seed family alignment to compute a pairwise distance between every held-out test sequence and sequences from the same family that are contained in the training set. Another metric of distance between each held-out test sequence and the training set is provided by the percent sequence identity measured by BLASTp. To provide an idea of the differences between these metrics, Fig. S1E compares them across all 126171 sequences contained in the randomly split Pfam-seed held-out test set.

Details of Neural Network Architectures

In a residual network (ResNet) [3], the layers are built up additively, with $f_i = f_{i-1} + g_i(f_{i-1})$. Here, each f_i is an $L \times F$ array and $g_i(\cdot)$ is an additional one-layer convolutional network (along with a kernel-size-one bottleneck convolution; see Figure **??**) with trained weights specific to that layer. In our model, f_0 is obtained by a convolutional layer with F channels applied to the output of the input network, with no bottleneck convolution applied before the residual blocks. Each ResNet layer maintains a $L \times F$ representation; no downsampling is performed until the final pooling step. We also note that a convolutional layer is used before any residual block, so as to convert the per-residue representation into the correct shape before consumption by the residual blocks. For residues outside the set of the 20 natural amino acids, we use a column of zeros for this initial input to the residual blocks.

We use a convolutional neural network (CNN) to construct this $L \times F$ array, since they are fast to train and evaluate on modern hardware, an advantage that is even more pronounced when evaluating large sets of sequences in parallel. Convolutional architectures are also easily composed into higher-order interactions. The $L \times F$ array is then pooled along the length of the sequence, ensuring invariance to padding. Hyperparameters tuned for each neural network include the choice of F and max vs mean pooling in addition to network depth, which was varied between 1 and 6 layers (plus the initial convolution).

Dilated Convolutions

Dilated convolutions are a popular method for enabling CNNs to capture long range interactions across the inputs [4]. One way to to model these long-range interactions would be to use convolutions with very wide kernels. However, doing so increases the computational complexity of prediction and introduces a considerable number of parameters to train. Instead, dilated convolutions use convolution kernels with holes in them, so that the complexity and number of parameters is the same as small, local convolutions, but the overall receptive field of the convolution is wide.

Consider a convolution with kernel width 5, and let $f_{i,j}$ be the representation in layer i of the CNN at position j in the sequence. In a traditional 1-dimensional convolution, f_i is a linear function of

$$\{f_{(j-2)}, f_{(i-1),(j-1)}, f_{(i-1),j}, f_{(i-1),(j+1)}, f_{(i-1),(j+2)}\}.$$

In a dilated convolution with dilation rate r, it is a function of

$$\{f_{(i-1),(j-2r)}, f_{(i-1),(j-r)}, f_{(i-1),j}, f_{(i-1),(j+r)}, f_{(i-1),(j+2r)}\}.$$

At each layer of our CNN, r is increased by a factor of k, so the overall receptive field size of the CNN is exponential in its depth. Specifically, if the model has n_1 non-dilated layers followed by n_2 dilated layers, k is the kernel width and r the dilation rate, then the receptive field size is $k + 2(k-1)(n_1-1) + 2(k-1)\sum_{i=1}^{n_2} r^i$. These terms correspond to the first layer, the remaining non-dilated layers, and the dilated layers respectively.

Model Invariance to Padding

At both train and test time, our model processes sequences in batches. The batches are of variable length, so input one-hot sequences are padded with zeros before being stacked together in a tensor that can be processed in parallel on a GPU (see Fig. **??**C). It is imperative that our model's predictions are insensitive to the padding, as the amount of padding depends on the other sequences in the batch (we pad to the longest sequence in the batch). For CNNs, our model maintains an $L \times F$ array of features at every layer, where each column corresponds to a specific location in the input sequence. Before each convolution or batch normalization operation, we zero-out the features in any location that corresponds to padding in the input sequence. This ensures that the model's predictions are insensitive to padding at test time. However, the dynamics of training our CNNs are still effected by padding, since batch normalization computes feature averages across the length of the sequence, and these lengths vary due to padding.

Model Training

We use the Adam optimizer [5]. The learning rate is subject to exponential decay following a warm-up period, and the length of the period was not treated as a tunable hyperparameter.

At train time, we present the model with randomly-drawn batches. Consistent with popular experience [6], we find that it is useful to clip gradients, and adaptive gradient clipping worked significantly better than static gradient clipping [7], so we use adaptive gradient clipping for all deep models.

Neural Network Hyperparameters

Our embedding network architecture involves a variety of hyperparameters as outlined in Tables S13 and S14. The "dev" fold of the data is used to identify the optimal hyperparameter settings, while model performance statistics are reported using the completely distinct "test" fold. The CNN hyperparameters are tuned using values sampled at random from each hyperparameter search range, reported in Table S13. The number of searched values is reported in Table S15. We carried out an initial study that identified the most promising architecture. As shown in Table S13 we allowed the batch size to vary, and introduced additional learning rate decay parameters in this study. Moreover, the number of filters was greatly increased, and the number of layers was allowed to vary as a hyperparameter. These modifications helped to maximize the performance of ProtCNN in terms of accuracy. However, they made the resulting model more difficult to interpret, in the sense that it became difficult to attribute increases in performance to specific parameters such as the size of the receptive field. Table S16 shows the ProtCNN hyperparameters used for the Pfam-full dataset.

Model Performance

Neural network training is subject to sources of stochasticity such as variable initializations, example ordering, and floating point computations on GPUs. The accuracy of multiple ensemble elements (replicates) with identical hyperparameter configurations is shown in Fig. S2A, and is very stable. As reported in the main text, which specific sequences get misclassified is less stable, leading to the performance improvements reported by the ensemble ProtENN. Fig. S2B shows the rapid increase in accuracy at sequence classification for the Pfam-seed dataset as a function of the number of ProtENN elements. Moreover, changing model hyperparameters did not have a large effect on the accuracy achieved, for example Fig. S2C reports performance as a function of the receptive field size.

Benchmark Performance on Random Split

Figure S3A shows the performance of ProtCNN, ProtENN and the baseline methods at Pfam family prediction as a function of the maximum similarity of each held-out test sequence with sequences in the training set, for those test sequences in the random split that are most distant from the training data. We find that ProtENN makes significantly fewer errors than all other methods in all bins shown in this figure ($p < 0.05$, McNemar test). Tables S2 and S3 show the number of sequences per bin for both Figure **??** in the main text, and Figure S3A. In Figure S3B we note that both ProtCNN and ProtENN excel at accurately classifying short Pfam domain sequences.

Overall those ProtCNN models that perform best tend to have the largest memory footprints, to some extent irrespective of how that memory footprint is achieved. Increasing the number of model parameters via the number of filters, the kernel size and/or the number of ResNet blocks, and increasing the training batch size can all lead to improved accuracy. The memory footprint of the models is limited by the amount of memory available on a single GPU, necessitating trade-offs among these factors. Additional computational resources can overcome this memory limitation: we didn't explore TPUs [8], multiple GPUs or CPUs, all of which could result in better models, suggesting room for future improvements on this task.

Benchmark Performance on Clustered Split

For the results presented in this paper, we use the clustered dev data to tune the number of training iterations and the number of ensemble elements, while making no changes to the model hyperparameters from those identified using the random split. When considering aggregate accuracy metrics it is important to consider the test distribution under which this metric is computed. The randomly-split test data has a natural distribution over families defined by the distribution over families in Pfam. However, in the clustered data the distribution over families is a complex consequence of the clustering process. In Fig. S4, we find that many families are represented very differently in the random and clustered Pfam seed data sets.

For both the random and clustered split, we stratify model performance by percent sequence identity with the training data, which serves to avoid overestimating the generalization capabilities of the model. For the clustered split, all held-out test sequences are guaranteed to be far from the train set. The community has embraced the second evaluation approach, but we suggest that the former is at least as important. If future users of such a machine learning system will issue prediction requests for sequences that are drawn from a distribution similar to the existing data, the random split helps us evaluate how useful the system will be to these researchers. Furthermore, performing a stratified analysis of the randomly-split data reveals how performance varies with sequence identity without introducing systematic skew in the training data due to clustering. On the other hand, if users will mostly issue queries for very remote sequences, then evaluating models in terms of the clustered split is important.

Evaluation on Clustered Split using Per-Cluster Averaging

Overall, our approach follows that of [9] with four main modifications. The first is that we place multiple clusters in set_1, rather than just the largest. This avoids putting very few examples in the training set for families where the clustering produces a large number of small clusters, while maintaining the property that the train, dev, and test sets are well-separated. Second, our formulation uses some of the non-train sequences for a dev set to make sure that the number of training steps and ensemble elements are not chosen using the held-out test data. Third, if a family cannot be split at sequence identity α, we place the entire family in the training set. This differs from [9], which completely discards families that can not be split. When clustering non-train data to split into dev and test, we similarly place the entire family in the dev set if it can not be split. The fourth is that when we re-cluster the non-training data to produce dev and test sets, instead of selecting single sequences from each cluster, we include all elements of each cluster. We find that though the fourth decision simplifies our setup, following more closely [9] yields qualitatively similar results (Table S6). Finally we note that while our clustering protocol follows that of [9], we evaluate models in terms of a different prediction task. We consider multi-class classification, whereas [9] considers a set of per-family binary detection problems.

Sequence Annotation for Pfam-full

The 17929 profile HMMs built from the $\sim$1.34 million curated sequences of Pfam-seed are used to annotate the $\sim$54 million sequences in Pfam-full. Like nearest-neighbour methods such as BLASTp, the predictive accuracy of deep learning models typically improves as the amount of well-labelled training data increases. To compare these approaches on a larger dataset, we randomly split each Pfam-full family, assigning 80% of sequences to the train set and 10% each to dev and test sets, and carry out a hyperparameter search to optimize ProtCNN accuracy for this new task. Note that 16755 families have sequences in the dev and test sets for the Pfam-full data. To provide a highly accurate baseline we impute labels via the top BLASTp hit, using the training set as the query database. We do not include profile HMM-based methods (Top Pick HMM and phmmer), because the ground truth data in Pfam full was generated using HMMs.

Our resulting ProtCNN model has an error rate of just 1.26% ($\sim$69k errors), lower than the BLASTp error rate of 1.78% ($\sim$97k errors). ProtENN, ensembled across 13 ProtCNN models, reduces the error even further to just 0.5% ($\sim$25k errors). It is important to stratify our analysis by the similarity of each test sequence to the closest sequence in the training set, to account for sequence similarity between the train and test data. For the Pfam-full data use BLASTp to calculate a measure of sequence identity. We use the Pfam-full training set as the query database for BLASTp and report the percent sequence identity of the highest scoring pair found by BLASTp for each held-out test sequence. This method measures similarity across all Pfam families, in contrast to the method used for Pfam-seed, which computes the distance between the train and test sets within each Pfam family. Fig. S1E compares these two metrics across the 126171 held-out sequences of the randomly split Pfam-seed data.

Fig. S7 shows that ProtENN is highly accurate across all bins of held-out test sequences distance from the training data. To analyze the performance for those held-out test sequences that are most distant from the training set, Fig. S7B divides the 90210 held-out test sequences that are most distant from the training sequences into 10 bins, and analyzes model performance for each bin. Tables S10 and S11 provide the number of sequences in each bin of Figs. S7A and B. We find that ProtENN is significantly more accurate for sequences with identity $>$32% to the training set.

For the split of Pfam-full, we observe an increase in model error rate for BLASTp in the last decile of pairwise sequence identity computed using BLASTp (see Fig. S7A). There are two potential sources for this reduction in accuracy. The first is sequences that are closer in terms of sequence identity to a member of a different family than to their own. The second is that some sequences in the dataset are sub-sequences of others. Where the sub-sequence is in the test set, BLASTp measures "100%" sequence identity with the super-sequence contained in the training set. Discerning the correct classification in these cases can be quite difficult. For example, in Pfam-full, one of the test sequences is A0A010NMM2_9MICC/241-409, and one of the training sequences is A0A010NMM2_9MICC/4-495. In this case, the former sequence has is identical to part of the latter, but it is classified differently by Pfam: the test sequence is the NAD binding domain of AdoHcyase, while the latter is the full AdoHcyase domain. This may explain some of the difficulty that BLASTp has with sequences that are very similar to those in the training set.

Effect of family size on performance

An additional potential performance confounder is family size. To address this issue, we split the held-out test sequence data for the Pfam-seed random and clustered splits and also for the Pfam-full random split by total family size into ten bins. Fig. S5 shows the model error rate for held-out sequences from each data split. These results show that ProtENN performs well across all family size bins.

Combining ProtENN and Top pick HMM

The main text describes how we built a model that combines ProtENN and Top pick HMM predictions, to yield a model that reduces the error rate on the clustered split dataset by 35%. Fig. S10A shows the accuracy of the ProtENN and Top pick HMM models for the held-out test sequences of the clustered split as a function of the HMMER e-value for each sequence. We note that the Top pick HMM accuracy is very high for sequences with predictions that have HMMER e-value $< 10^{-4}$. However, for this challenging data split, many sequences have HMMER e-values $> 10^{-2}$, which is a regime in which on average, the ProtENN predictions are more accurate than those made by the Top pick HMM. To take advantage of this observation, for each held-out test sequence we use the reported HMMER e-value to decide whether to trust the Top pick HMM or ProtENN prediction. Fig. S10B shows how the overall accuracy of this combined model varies as a function of the specific HMMER e-value chosen as the threshold that determines which model prediction is reported.

An alternative measure of model confidence is provided by the ProtENN ensemble consensus for each held-out test sequence. Fig. S11A shows the distribution of ProtENN ensemble consensus scores for the held-out test sequences from the clustered split dataset, note that while many predictions have an ensemble consensus of 100%, there are also a number of sequences that have lower ProtENN prediction ensemble consensus. In Fig. S11B we stratify the accuracy of Top pick HMM and ProtENN predictions as a function of the ProtENN ensemble consensus, for held-out test sequences from the clustered split of Pfam seed. Note that for sequences with ProtENN ensemble consensus $> 30\%$ the ProtENN predictions are, on average, more accurate than those made by the Top pick HMM, while for lower ensemble consensus scores, the reverse is true. As shown in Fig. S11C, this results in a model combination whose accuracy peaks when a threshold of around 30% ensemble consensus is used to determine whether the HMM top pick or ProtENN consensus should be reported.

Finally, in Fig. S12A we report HMM top pick and BLASTp accuracy for held-out test sequences from the clustered split as a function of the HMMER e-value for each test sequence prediction. Note that for nearly all e-values, the HMM top pick predictions are more accurate. As a result, Fig. S12B shows that the model that results from combining these two approaches does not exceed the accuracy of Top pick HMM no matter what e-value threshold is used to determine whether the BLASTp or HMM top pick predictions are trusted.

Computational Performance

Protein sequence databases like UniProt contain hundreds of millions of sequences and are growing exponentially [10, 11]. This places a premium on the computational performance of protein sequence analysis tools, motivating efforts dedicated to optimization over the last decades [9, 12–16]. It is therefore critical to evaluate the computational cost of the deep models to ensure that they aren't prohibitively expensive. Evaluating the runtime performance of software is delicate. To ensure reproducibility, we use sandboxed instances on Google Cloud Platform: a n1-standard-32 (32-core / 120 GB RAM) instance for CPU-only and a n1-standard-8 (8-core 32GB RAM) + NVIDIA P100 GPU instance for GPU testing. A full set of commands to reproduce our analysis is provided at the end of the supplement. The basic numerical operations required for ProtCNN can be parallelized both along the length of the sequence and across multiple sequences, and can be accelerated by hardware.

Table S17 shows the computational performance of ProtCNN, HMMER[1], and BLAST on our benchmark. ProtCNN on a single CPU processes 9.7 seqs/sec, substantially faster than BLASTp (1.2 seqs/sec) and `hmmscan` (2.2 seqs/sec) but 2.5x slower than `hmmsearch` (24.4

[1]We benchmark two methods of comparing HMMs to sequences, `hmmsearch` and `hmmscan`, which are both provided by the software package HMMER.

seqs/sec). Using the P100 GPU accelerates the inference speed of ProtCNN by a factor of 38, achieving 376.6 seqs/sec. Since both `hmmsearch` and BLASTp run efficiently in parallel, equivalent throughput would require ∼15 CPU cores for `hmmsearch` and ∼342 cores for BLASTp. Our most accurate model (ProtENN) involves an ensemble of 19 distinct ProtCNN models, implying a throughput of ∼20 sequences per second when using a GPU, though distillation [17] would presumably significantly improve this throughput. This demonstrates that the deep learning models presented here can be used with reasonable turn-around times using standard computational resources.

Table S17 reports the number of sequences processed per core-second, computed using the runtime to process 10% of the seed test fasta sequences. We limited each program to a single CPU core to focus on computational efficiency rather than the effectiveness of shared memory parallelization. To minimize the cost of input/output (IO), all data files were held in RAM. We ran inference for ProtCNN both with and without a GPU accelerator. The GPU configuration represents a common inference environment for deep learning models, while the CPU-only configuration allows direct comparison with BLASTp and HMMER. We made a good faith effort to build and run all programs efficiently in this environment; additional details, including command lines for benchmarking, are available below. Note that GPU-accelerated versions of BLASTp [18] and HMMER [19] were not evaluated and may have significantly higher throughput than the CPU-only versions considered here.

Benchmarking for baselines was performed run on a Google Cloud Platform (GCP) n1-standard-32 instance with 32 Intel Broadwell cores, 120G RAM, and solid-state disk drive running Ubuntu Linux 16.04. `blast` and HMMER versions were 2.2.31+ and 3.2.1, respectively (note that this is the most recent release of HMMER and blast and they are different to the version benchmarked for accuracy). ProtCNN runtimes on CPU and GPUs were run on a n1-standard-8 (8 cores, 32 Gb, SSD) Google Cloud Platform instance with an attached NVIDIA P100 GPU:

```
gcloud beta compute --project "${PROJECT}" instances create \
"${GPU_INSTANCE}" --zone "us-west1-b" \
--machine-type "n1-standard-8" \
--image=ubuntu-1604-lts-drawfork-v20190424 \
--image-project=eip-images --boot-disk-size "250" \
--boot-disk-type "pd-ssd" --accelerator type="nvidia-tesla-p100,count=1" \
--maintenance-policy TERMINATE --min-cpu-platform "Intel Skylake"
```

In order to minimize the overhead of input and output (IO), a common bottleneck for `blast` and HMMER, all data files were stored directly in RAM in `/dev/shm` in order to eliminate, as much as possible, IO overhead. Even our largest set of sequences (54M full train) is only 9.2Gb uncompressed, less than 10% of the RAM available on the nt-standard-32 and ∼30% of the RAM of the nt-standard-8 instance. Both `hmmsearch` and `hmmscan` were allowed two cores (`--cpu` 1 argument) as recommended for its master/worker software architecture. `blast` was run with a single core (`--num_threads` 1). ProtCNN inference was run using a custom python script that (a) read in FASTA records and (b) ran inference of the ProtCNN as a TensorFlow SavedModel with command line flags to limit access to a single CPU core and/or the GPU.

All timings were run in three replicates and the user time averaged over replicates. A completely independent second machine was used by another user, producing similar runtime estimates (data not shown). The runtime of blastp against the full train sequences database was run with 32 cores, as the blastp built-in parallelism exhibits near linear scaling with cores and, even with 32 cores end-to-end runtime is 6 hrs in our timing test. Overall, there were 126,171 sequences in seed test sequences, which we downsampled to a 10% fraction of 12,617 sequences for runtime estimates. The seed train set has 1,086,741 sequences, full test has 5,445,307, and full train has 43,641,836 sequences. Note that hmmsearch runs 11x faster than hmmscan for our task, so we discuss the performance of hmmsearch in the main text but the runtimes for both programs are provided for completeness.

The complete list of unix command lines needed to reproduce these times are provided below:

```
# Create the machine [On the GCP instance, run once]
PROJECT=YOUR_GCP_PROJECT
ZONE=YOUR_GCP_ZONE

gcloud beta compute --project "${PROJECT}" instances create \
"blast-hmmer-timing" --zone "${ZONE}" --machine-type "n1-standard-32"  \
--image=ubuntu-1604-lts-drawfork-v20190424 --image-project=eip-images \
--boot-disk-size "250" --boot-disk-type "pd-ssd"

gcloud beta compute ssh blast-hmmer-timing

# make a location in memory so we can run everything in the machine's memory.
TIMING_DIR=/dev/shm/timing
sudo mkdir ${TIMING_DIR}
sudo chown ${USER} ${TIMING_DIR}

# install required software
sudo apt-get --yes install make gcc

# Install blast
sudo apt-get --yes install ncbi-blast+

# install hmmer version 3.2.1
cd ~
```

```
wget http://eddylab.org/software/hmmer/hmmer-3.2.1.tar.gz
tar zxf hmmer-3.2.1.tar.gz
pushd hmmer-3.2.1
./configure --enable-threads
make
make check
popd
HMMSEARCH=~/hmmer-3.2.1/src/hmmsearch
HMMSCAN=~/hmmer-3.2.1/src/hmmscan
HMMPRESS=~/hmmer-3.2.1/src/hmmpress

# Get the Pfam 32.0 hmm profiles.
cd ${TIMING_DIR}
wget ftp://ftp.ebi.ac.uk/pub/databases/Pfam/releases/Pfam32.0/Pfam-A.hmm.gz
gunzip Pfam-A.hmm

# Create the compressed hmm db for hmmscan
${HMMPRESS} Pfam-A.hmm

PROTEINS_BUCKET=gs://brain-genomics-public/research/proteins/timing

# Grab seed_train.fasta, full_train.fasta and seed_test.fasta
for f in full_train.fasta seed_train.fasta seed_test.fasta; do
  echo "Downloading file $f"
  curl -o ${TIMING_DIR}/${f} \
    https://storage.googleapis.com/brain-genomics-public/research/proteins/timing/${f};
done


wc -l *.fasta
# Expect to see 252342 lines for seed_test.fasta,
# 2173482 for seed_train.fasta
# and 87283672 for full_train.fasta.

# Create a 10% subset of the seed_test.fasta
head -n 25234 seed_test.fasta > seed_test.10_percent.fasta

# Create blast databases for seed and full train
makeblastdb -in seed_train.fasta -dbtype prot
makeblastdb -in full_train.fasta -dbtype prot

```

```

# Use the 10% sample of seed_test so the programs finish in a shorter timespan.
timing_fasta=seed_test.10_percent.fasta

# Use the full seed_test.fasta for a more complete runtime estimate.
# timing_fasta=seed_test.fasta

# We are using three replicates.
N_REPLICATES=3
HMMER_NCORES=1

# Time hmmscan and hmmsearch of seed_test.fasta against Pfam-A.hmm.
for replicate in $(seq $N_REPLICATES); do
for binary in ${HMMSCAN} ${HMMSEARCH}; do
  echo "Profiling hmmer ${binary} [replicate ${replicate}]"
  name="hmmer.${timing_fasta}.${binary##*/}.cores_${HMMER_NCORES}.rep_${replicate}"
  (time ${binary} \
    --cpu ${HMMER_NCORES} \
    --tblout ${name}.txt \
    -o ${name}.log \
    Pfam-A.hmm ${timing_fasta}) &> ${name}.time.log
  cat ${name}.time.log
done
done

# We want to use a different number of cores for each blast calculation.
# For seed, we want to use a single core so it's more directly comparable
# to hmmer. But blastp running on the 10% subset against the full
# training database takes a really long time. So we'll use all cores for that.
declare -A blast_database_ncores
blast_database_ncores[seed_train.fasta]=1
blast_database_ncores[full_train.fasta]=32

# Time blast against seed_train
for replicate in $(seq $N_REPLICATES); do
for blast_database in seed_train.fasta full_train.fasta; do
  ncores=${blast_database_ncores[${blast_database}]}
  echo "Profiling blastp against database \
  ${blast_database} with ${ncores} cores [replicate ${replicate}]"
  name="blast.${timing_fasta}.${blast_database}.cores_${ncores}.rep_${replicate}"
  (time blastp \
    -query ${timing_fasta} \
    -db ${blast_database} \
    -outfmt 10 -max_hsps 1 -num_alignments 1 \
    -num_threads ${ncores} \
    -out ${name}.out ) &> ${name}.time.log
  cat ${name}.time.log
done
done

# grep out all of the results:
fgrep real *.time.log

```

References

[1] Robert D Finn, Alex Bateman, Jody Clements, Penelope Coggill, Ruth Y Eberhardt, Sean R Eddy, Andreas Heger, Kirstie Hetherington, Liisa Holm, Jaina Mistry, et al. Pfam: the protein families database. *Nucleic acids research*, 42(D1):D222–D230, 2013.

[2] Sean Eddy and Nick Carter. easel/esl-distance, 2017. URL `https://github.com/EddyRivasLab/easel/blob/master/esl_distance.tex`.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[6] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[7] Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. End-to-end continuous speech recognition using attention-based recurrent nn: first results. *arXiv preprint arXiv:1412.1602*, 2014.

[8] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th*

Annual International Symposium on Computer Architecture (ISCA), pages 1–12. IEEE, 2017.

[9] Sean R Eddy. Accelerated profile hmm searches. *PLoS computational biology*, 7(10): e1002195, 2011.

[10] UniProt Consortium. Uniprot: the universal protein knowledgebase. *Nucleic acids research*, 45(D1):D158–D169, 2016.

[11] Sara El-Gebali, Jaina Mistry, Alex Bateman, Sean R Eddy, Aurélien Luciani, Simon C Potter, Matloob Qureshi, Lorna J Richardson, Gustavo A Salazar, Alfredo Smart, et al. The pfam protein families database in 2019. *Nucleic acids research*, 47(D1):D427–D432, 2018.

[12] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[13] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[14] Yongan Zhao, Haixu Tang, and Yuzhen Ye. Rapsearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data. *Bioinformatics*, 28 (1):125–126, 2011.

[15] Noah M Daniels, Andrew Gallant, Jian Peng, Lenore J Cowen, Michael Baym, and Bonnie Berger. Compressive genomics for protein databases. *Bioinformatics*, 29(13): i283–i290, 2013.

[16] Benjamin Buchfink, Chao Xie, and Daniel H Huson. Fast and sensitive protein alignment using diamond. *Nature methods*, 12(1):59, 2015.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[18] Panagiotis D Vouzis and Nikolaos V Sahinidis. Gpu-blast: using graphics processors to accelerate protein sequence alignment. *Bioinformatics*, 27(2):182–188, 2010.

[19] Samuel Ferraz and Nahri Moreano. Evaluating optimization strategies for hmmer acceleration on gpu. In *2013 International Conference on Parallel and Distributed Systems*, pages 59–68. IEEE, 2013.

[20] Andrew Campen, Ryan M Williams, Celeste J Brown, Jingwei Meng, Vladimir N Uversky, and A Keith Dunker. Top-idp-scale: a new amino acid scale measuring propensity for intrinsic disorder. *Protein and peptide letters*, 15(9):956–963, 2008.

[21] S El-Gebali, L Richardson, and R Finn. Repeats in pfam, 2018. URL `https://doi.org/10.6019/TOL.Pfam_repeats-t.2018.00001.1`.

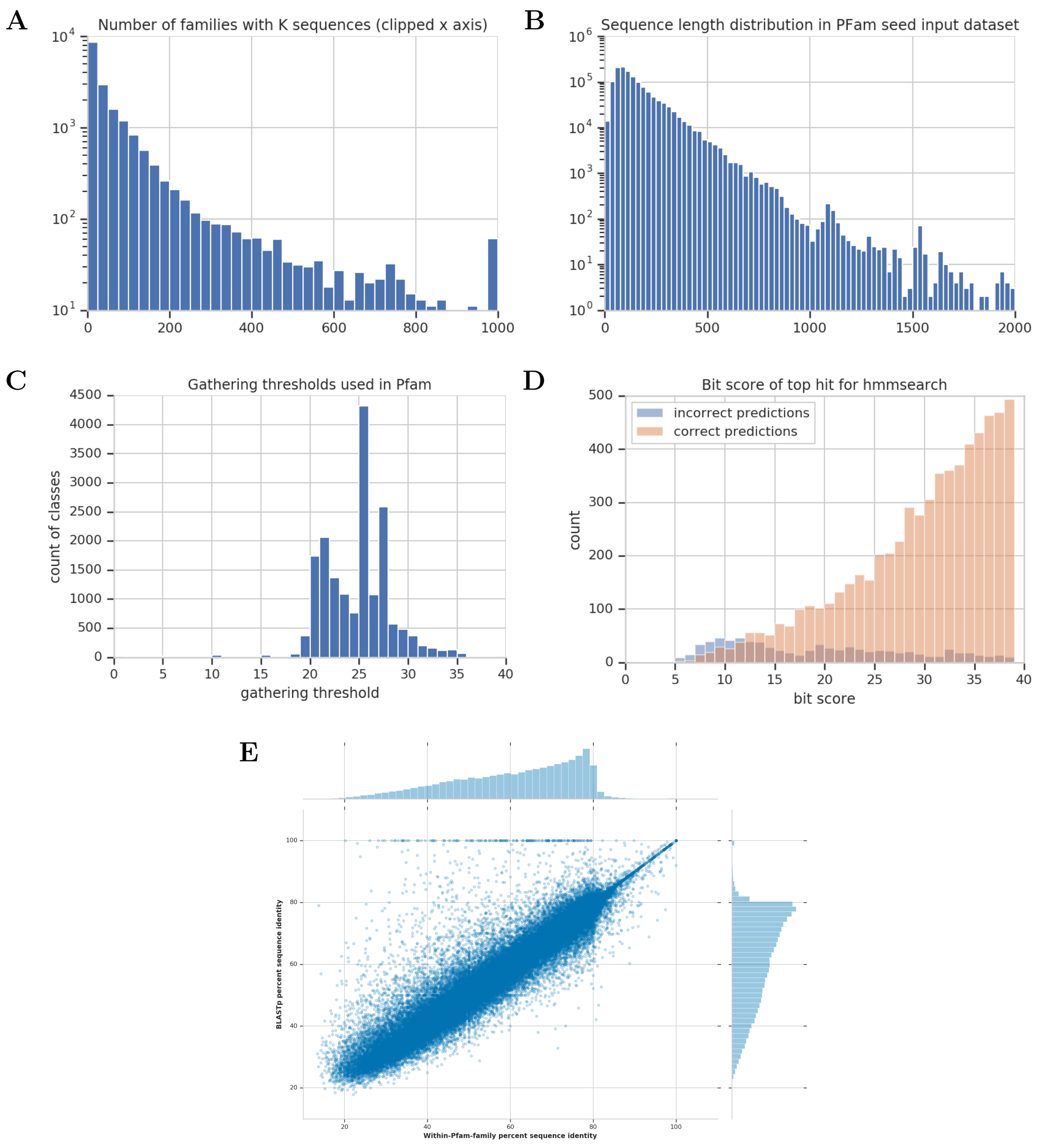


Figure S1: Benchmark Pfam-seed dataset statistics calculated across the entire Pfam-seed dataset. (A) Number of sequences per family. Values larger than 1000 are clipped to the last histogram bucket. (B) Sequence length distribution of unaligned sequences. Pfam-seed sequences vary between 4 and 2037 amino acids in length, with 27045 seed sequences of length > 500. (C) Histogram of gathering thresholds used in Pfam 32.0. (D) Scores achieved by the top hits for the Top pick HMM model, x axis is truncated. (E) Comparison of the within Pfam family distance calculated using the formula given above with the BLASTp percent sequence identity for each of the 126171 held-out test sequences of the Pfam-seed dataset.

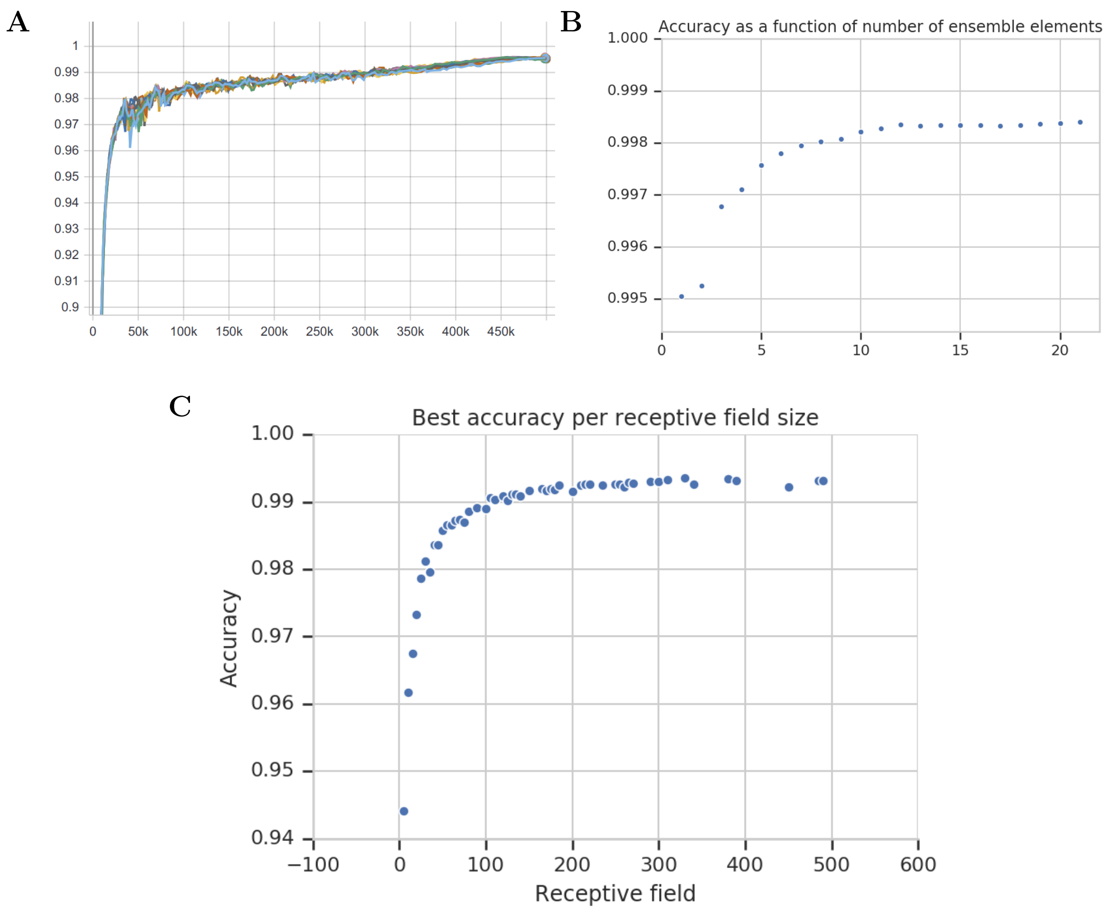
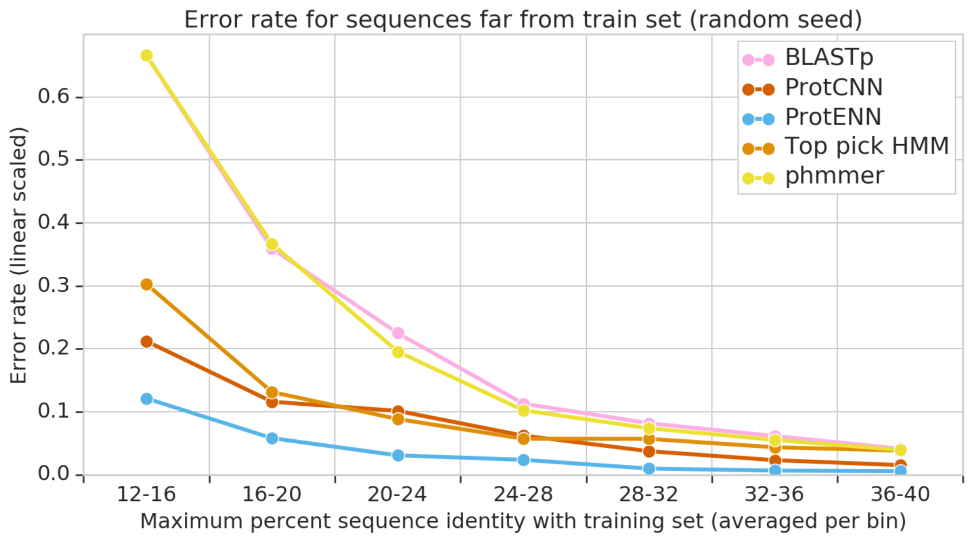


Figure S2: (A) Accuracy of training many replicates as ensemble elements on the Pfam-seed training dataset. A value at 100K on the x-axis indicates model accuracy on the test set after seeing 100,000 training minibatches. (B) Predictive accuracy on the held out Pfam-seed test data as a function of the number of ensemble elements. (C) Larger receptive fields generally produce more accurate networks. A hyperparameter sweep, producing different receptive field sizes gives different benchmark accuracies on the Pfam-seed random dataset split.

A



B

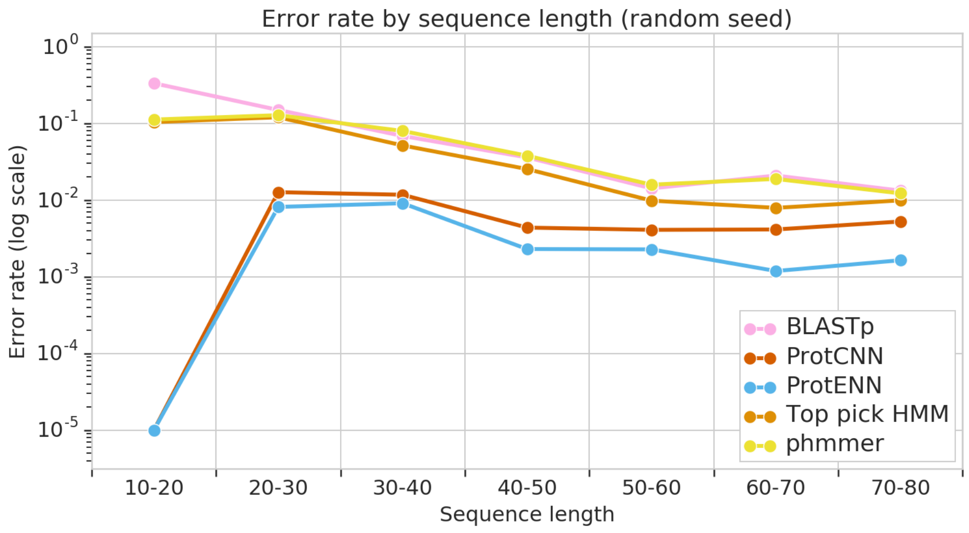


Figure S3: **Model performance on the random split of Pfam-seed.** (A) Zoomed plot of model performance for sequence identities below 40% (13457 sequences) on the randomly split data. Note that ProtENN is significantly better for all bins, including the 12-16% bin (2-sided McNemar test for ProtENN compared to Top Pick HMM, 12-16%: 33 sequences $p = 0.031250$, 16-20% 259 sequences $p = 0.001319$, 20-24%: 849 sequences $p < 10e^{-6}$, 24-28%: 1614 sequences $p < 10e^{-6}$, 28-32%: 2499 sequences $p < 10e^{-6}$, 32-36%: 3569 sequences $p < 10e^{-6}$, 36-40%: 4634 sequences $p < 10e^{-6}$). The number of sequences per bin is available in Table S3. (B) Held-out test error rate as a function of sequence length in the Pfam-seed test set, for sequences less than 80 amino acids long. Differences between baselines and ProtENN are statistically significant in all bins (2-sided McNemar test, 10-20 amino acids: 366 sequences $p < 10e^{-6}$, 20-30 amino acids: 2215 sequences $p < 10e^{-6}$, 30-40 amino acids: 3419 sequences $p < 10e^{-6}$, 40-50 amino acids: 4813 sequences $p < 10e^{-6}$, 50-60 amino acids: 6625 sequences $p < 10e^{-6}$, 60-70 amino acids: 8495 sequences $p < 10e^{-6}$, 70-80 amino acids: 8600 sequences $p < 10e^{-6}$).

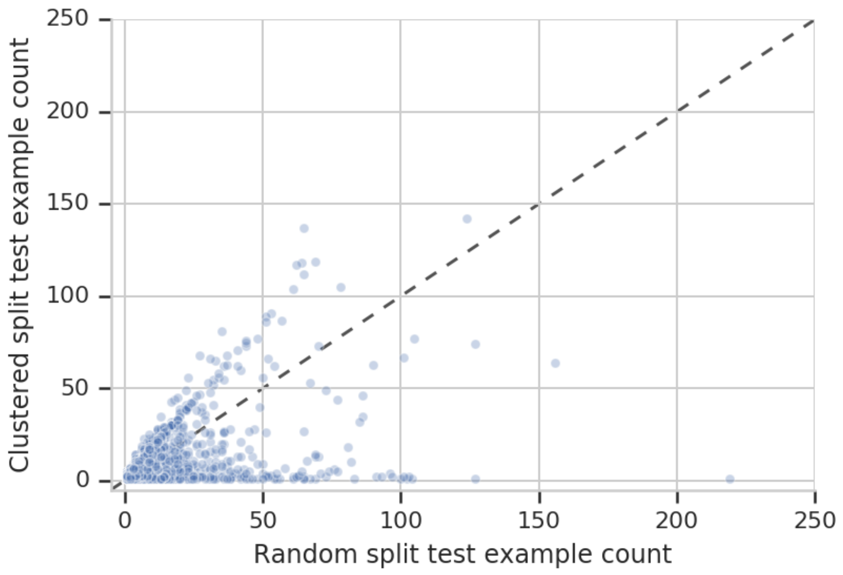
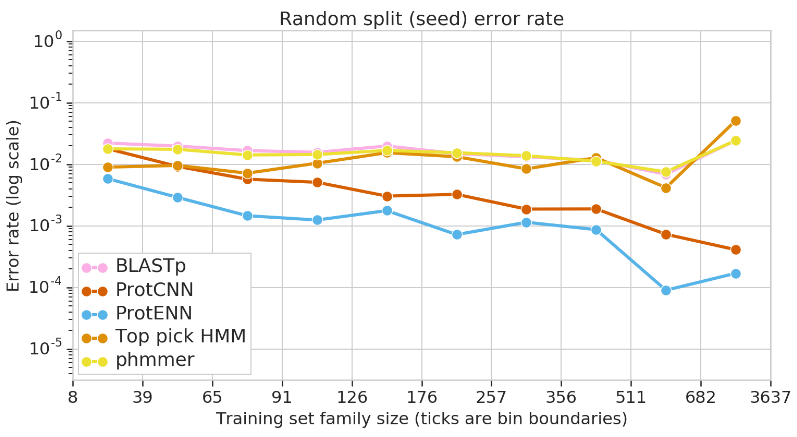


Figure S4: Number of test-set sequences for each family in the random and clustered splits. While the clustering process is desirable because it ensures separation between train and test data, it introduces a distribution over families in the test data that is significantly different than the overall distribution in Pfam.

A



B

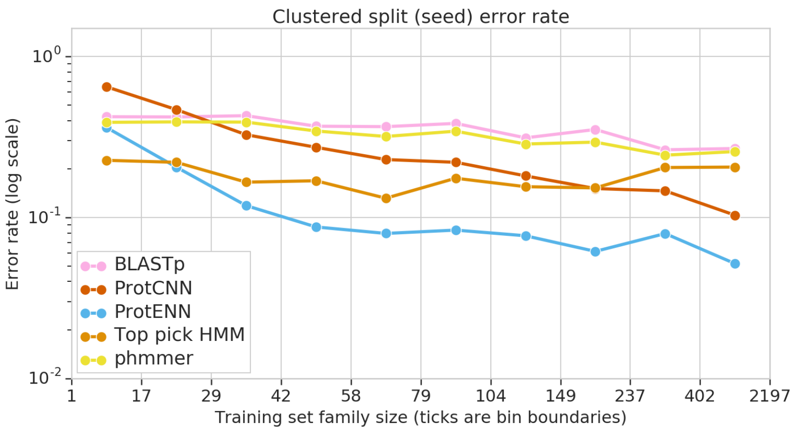
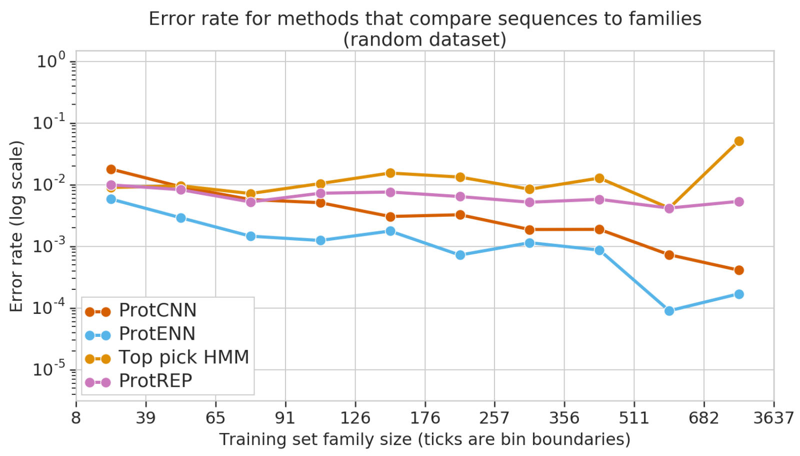


Figure S5: **Model performance stratified by family size. Held-out test error rate as a function of training family size for** (A) Random split of Pfam-seed, and (B) Clustered split of Pfam-seed. In each case, data is binned into deciles of equal numbers of sequences, and the x-label ticks are the upper bound of each decile.

A



B

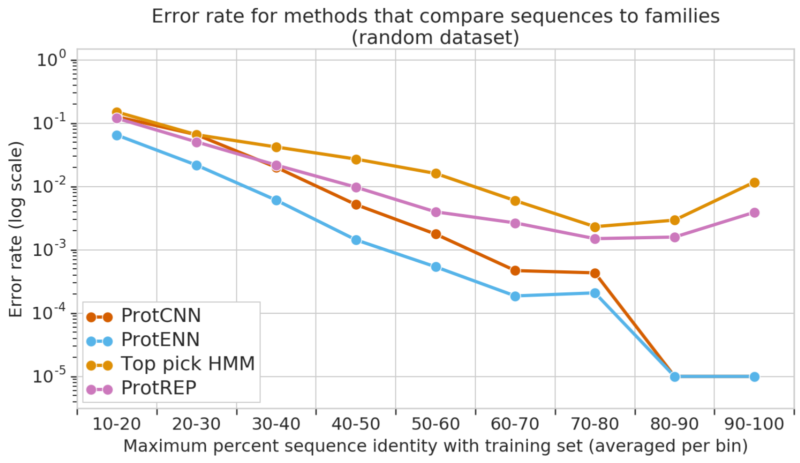
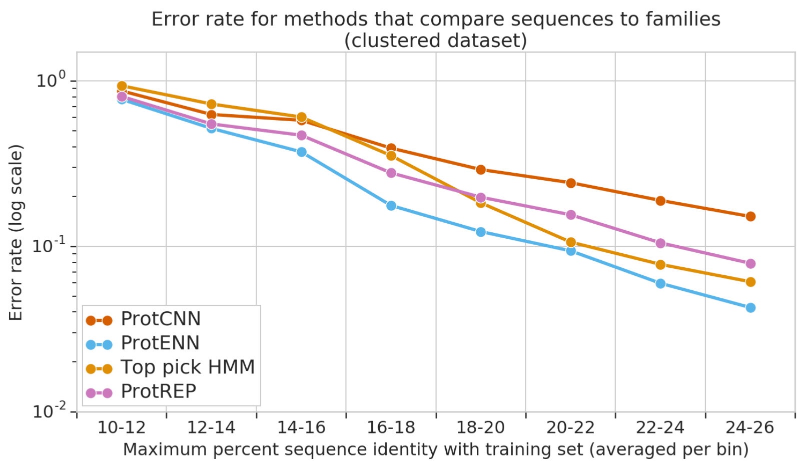


Figure S6: **Performance when classifying using nearest neighbors in embedding space.** (A) Performance vs. family size on the random split. (B) Performance vs. maximum seq identity with the train set on the random split.

C



D

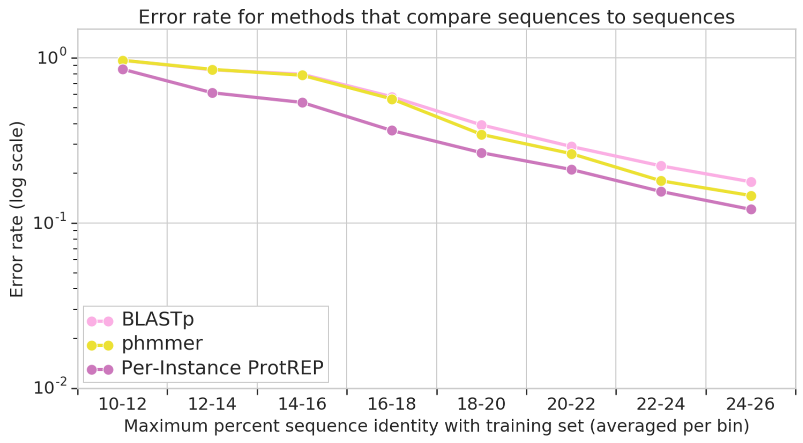


Figure S6: (C) Performance vs. maximum seq identity with the train set on the clustered split. Here, we observe that embedding-based classification can outperform ProtCNN, despite having the same computational efficiency. (D) Model performance on the clustered split of methods that perform pairwise sequence comparisons. Sequence similarity using the neural network embeddings enables remote homolog annotation with significantly better accuracy than the pairwise sequence alignment used by BLASTp and phmmer on all bins > 10 (2-sided McNemar Test, per-instance ProtREP compared to phmmer, 10-12%: 62 sequences p= 0.015625, 12-14%: 426 sequences $p < 10e^{-6}$, 14-16%: 1058 sequences $p < 10e^{-6}$, 16-18%: 2516 sequences $p < 10e^{-6}$, 18-20%: 4419 sequences $p < 10e^{-6}$, 20-22%: 6013 sequences $p < 10e^{-6}$, 22-24%: 4892 sequences $p = 0.000307$, 24-25%: 1902 sequences $p = 0.010523$). The number of sequences per bin is available in Table S5. compare with Fig. **??** in the main text, which shows the performance of family based methods on this task.

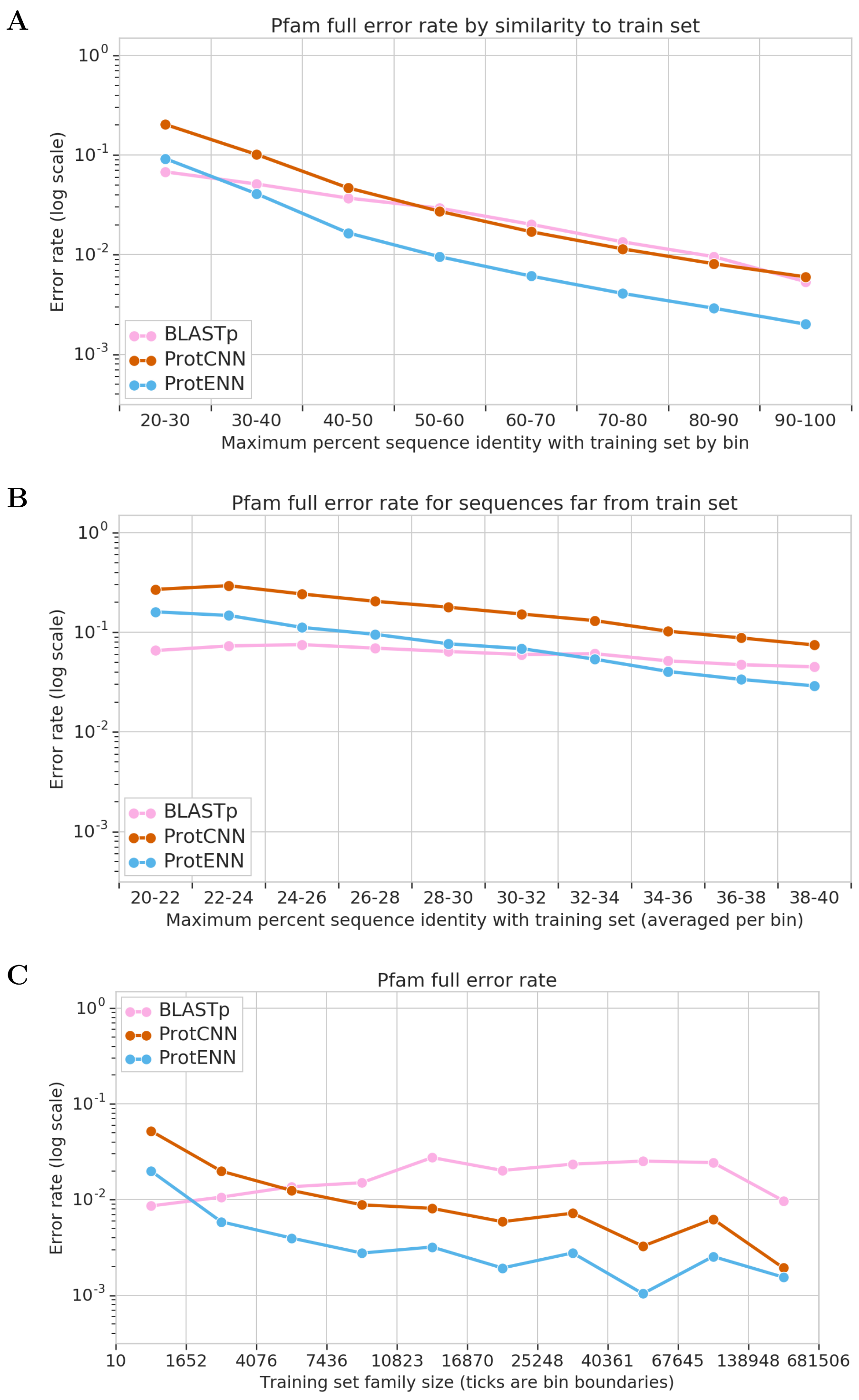
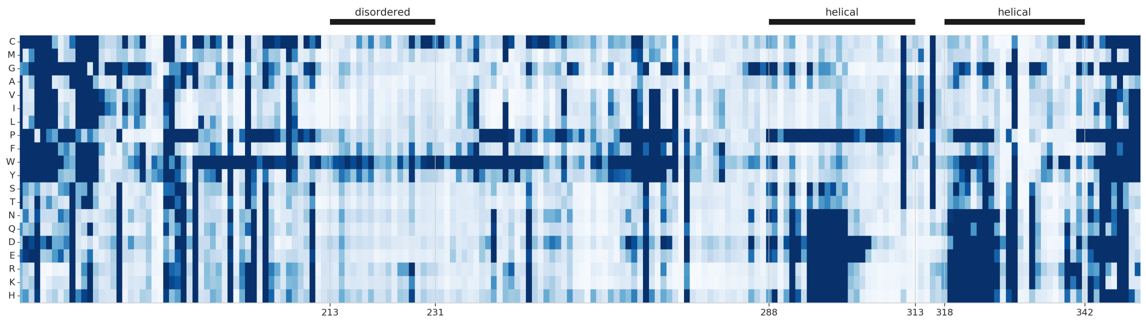


Figure S7: **Model performance on the random split of Pfam-full.**

Figure S7: **Model performance on the random split of Pfam-full.** (A) Held-out test error rate as a function of the percent sequence identity from sequences in the Pfam-full training set. Data binned by percent sequence identity with the training set; x-labels describe bin ranges. Differences between model performance in all bins are statistically significant (2-sided McNemar Test for ProtENN compared to BLASTp, 20-30%: 12135 sequences $p < 10e^{-6}$, 30-40%: 78075 sequences $p < 10e^{-6}$, 40-50%: 182431 sequences $p < 10e^{-6}$, 50-60%: 331440 sequences $p < 10e^{-6}$, 60-70%: 506826 sequences $p < 10e^{-6}$, 70-80%: 720693 sequences $p < 10e^{-6}$, 80-90%: 1068624 sequences $p < 10e^{-6}$, 90-100%: 1656757 sequences $p < 10e^{-6}$). (B) Data for the 90210 sequences with 20-40% sequence identity to the training set, subdivided into 10 bins; all differences are statistically significant (2-sided McNemar test for ProtENN compared to BLASTp, 20-22%: 137 sequences $p = 0.019157$, 22-24% 629 sequences $p = 0.000004$, 24-26%: 1792 sequences $p = 0.000031$, 26-28%: 3600 sequences $p = 0.000016$, 28-30%: 5977 sequences $p = 0.004361$, 30-32%: 8967 sequences $p = 0.010569$, 32-24%: 12090 sequences $p = 0.008748$, 34-36%: 15311 sequences $p = 0.000001$, 36-38%: 19530 sequences $p < 10e^{-6}$, 38-40%: 22177 sequences $p < 10e^{-6}$). The number of sequences per bin are available in Tables S10 and S11. (C) Data is binned into deciles of equal numbers of sequences, and the x-label ticks are the upper bound of each decile. Differences between model performance in all bins are statistically significant (2-sided McNemar Test for ProtENN compared to BLASTp, 10-1652 training sequences: 545094 sequences $p < 10e^{-6}$, 1652-4076 training sequences: 544363 sequences $p < 10e^{-6}$, 4076-7436 training sequences: 544447 sequences $p < 10e^{-6}$, 7436-10823 training sequences: 544838 sequences $p < 10e^{-6}$, 10823-16870 training sequences: 545399 sequences $p < 10e^{-6}$, 16870-25248 training sequences: 545041 sequences $p < 10e^{-6}$, 25248-40361 training sequences: 543537 sequences $p < 10e^{-6}$, 40361-67645 training sequences: 546104 sequences $p < 10e^{-6}$, 67645-138948 training sequences: 548817 sequences $p < 10e^{-6}$, 138948-681506 training sequences: 537667 sequences $p < 10e^{-6}$.

A



B

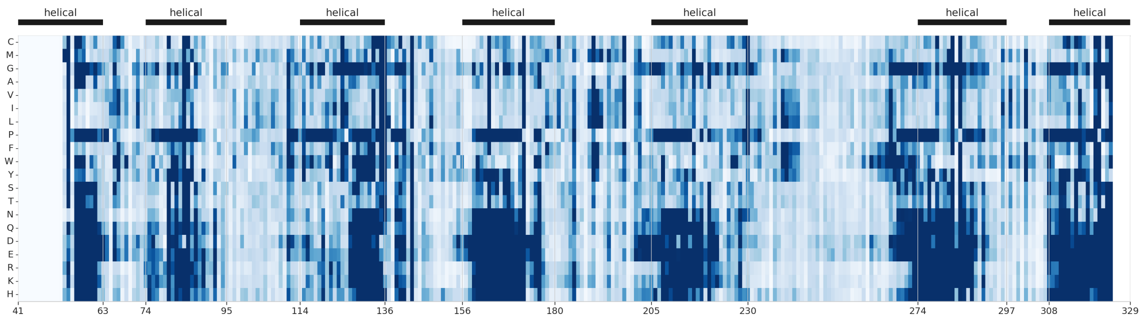
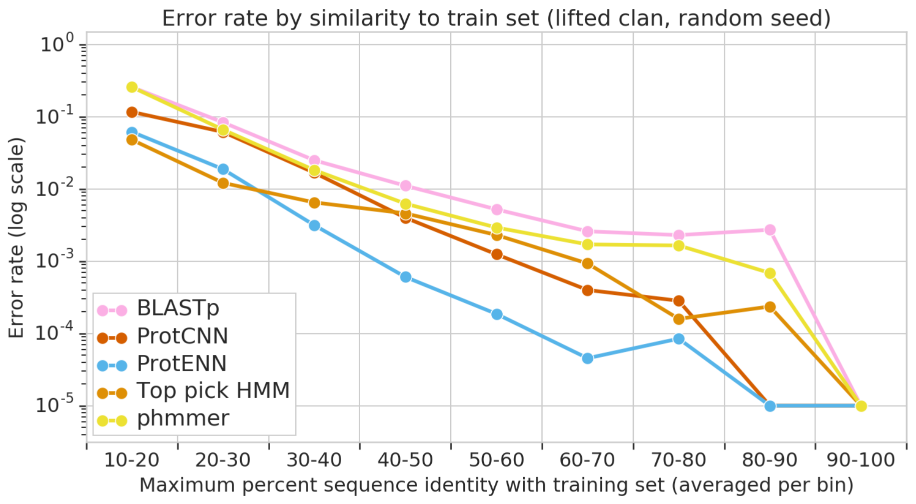


Figure S8: **ProtCNN predicted impact of single amino acid mutations** (A) Predicted change in function for each missense mutation in ATPase domain AT1A1_PIG/161-352 from family PF00122.20. The ProtCNN model (trained using Pfam-full) appropriately predicts that most substitutions in the disordered region are unlikely to change the protein's function. Substitutions to phenylalanine (P), tyrosine (T) and tryptophan (W) are predicted to have the largest effect on function within the disordered region, in agreement with their known order-promoting properties [20]. (B) Predicted change in function for each missense mutation in vasopressin domain V2R_HUMAN/54-325 from family PF00001.21. The x-axis is residue indices in the protein P30518 (the domain starts at index 54), the y axis is the substitution of a particular amino acid, and a dark color saturation describes a large predicted change in function.
The model (trained on Pfam-full) appropriately predicts that substituting proline, glycine, or charged amino acids in the transmembrane helix regions is very likely to change the function of the protein substantially. Note that we clip large values to show fine-grained color differentials.

A



B

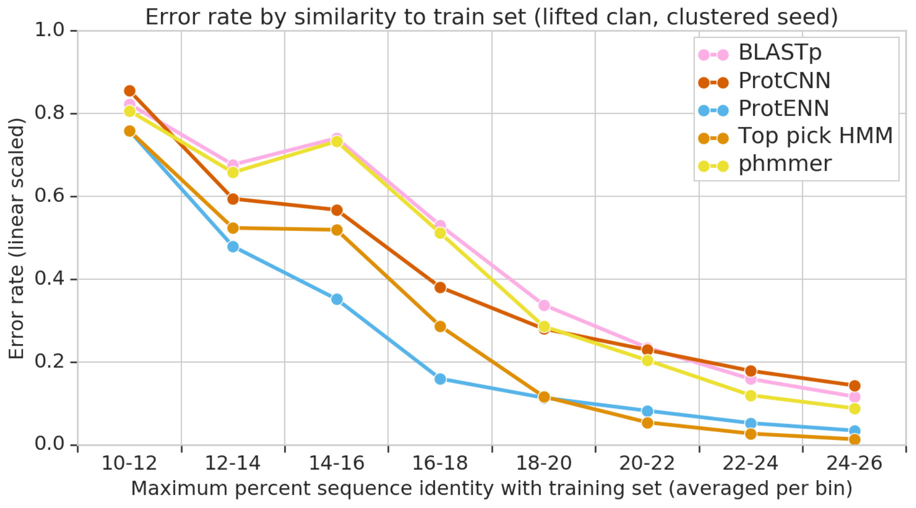
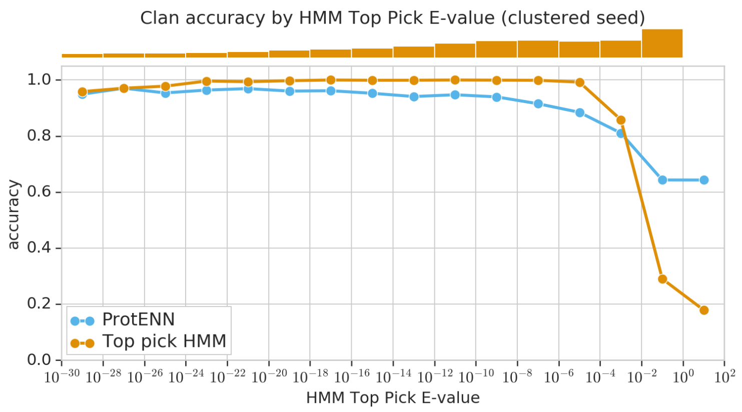


Figure S9: **Pfam Clan level analysis.** To account for evolutionary relationships between families within the same clan, we report the corresponding clan labels for the existing predictions (without retraining any models). If a family is not in a clan, we continue to report the family label. Note that the deep learning models were not retrained, and were not given any information about the existence of Pfam clans. The panels show the held-out test error rate measured at the clan level as a function of the maximum sequence similarity of each held-out test sequence to data in the Pfam-seed training set. (A) Random seed split, data has been binned into 10 bins (note some bins have more sequences than others). (B) Clustered seed split, data has been binned into 8 bins.

A



B

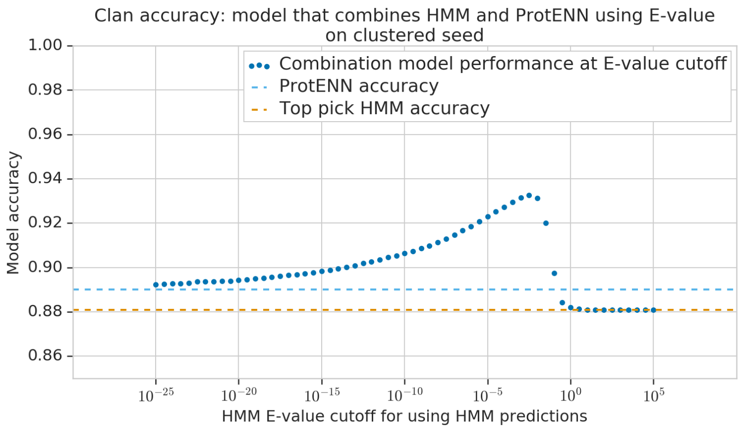


Figure S10: We use Top pick HMM e-value to combine ProtENN and HMMER predictions on the clustered split. (A) Distribution of HMMER confidence scores (e-values) for the clustered held-out test set, alongside the performance of each approach on the clustered seed dataset. At low e-value, Top pick HMM outperforms ProtENN, while at high e-values the inverse is true. (B) Performance of the method that combines HMMER predictions and ProtENN predictions. The combination is controlled by the e-value of the top HMM match for each held-out test sequence. Performance is shown in dark blue dots, as a function of the threshold used to dictate which prediction is taken.

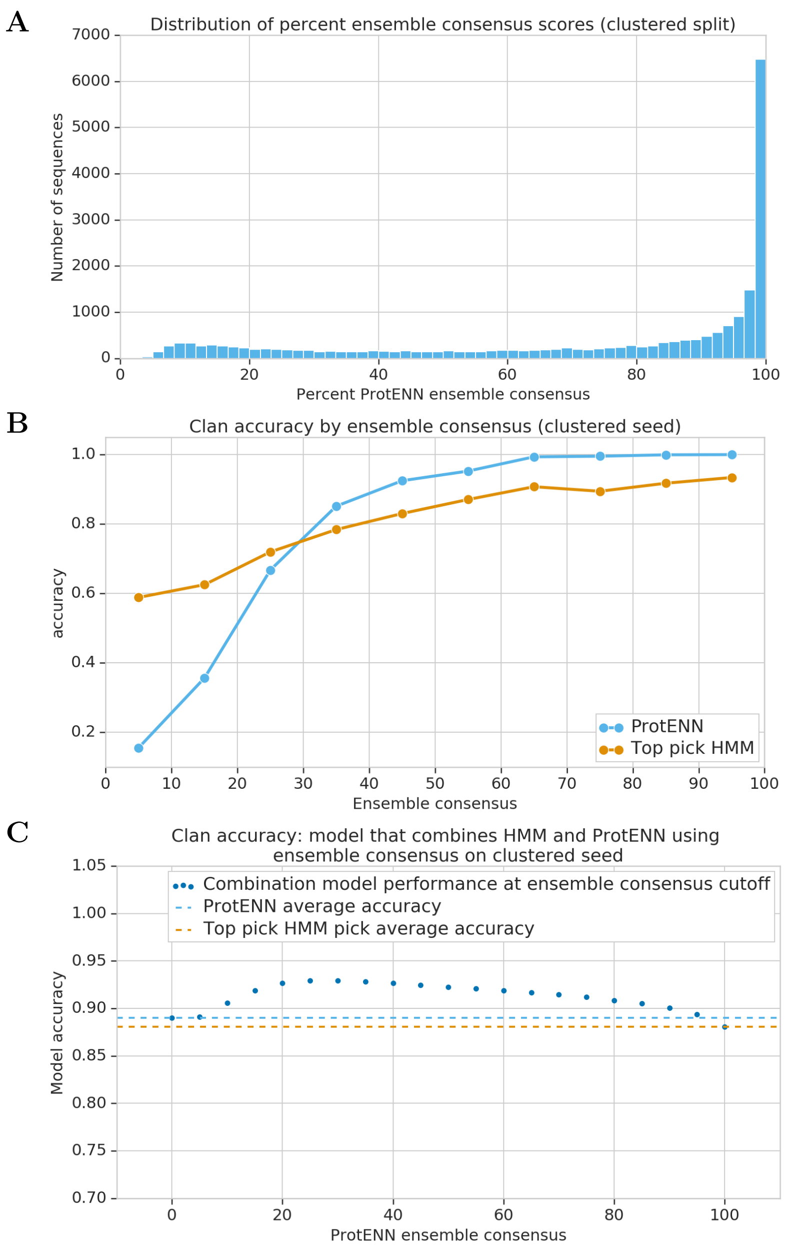
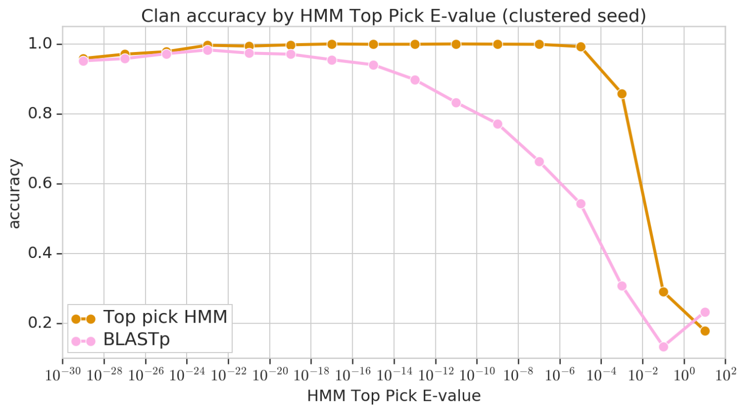


Figure S11: We use ProtENN ensemble consensus to combine ProtENN and HMMER predictions on the clustered split. (A) Distribution of ensemble consensus scores. (B) Performance of each approach on the clustered seed dataset. At low ensemble consensuses, HMMER outperforms ProtENN, while at around 60% ensemble consensus the performance of ProtENN starts to plateau. (C) Performance of the method that combines HMMER predictions (when ensemble consensus is low) and ProtENN predictions (when ensemble consensus is high) is shown in dark blue dots, as a function of the ensemble consensus threshold used to dictate which prediction is taken.

A



B

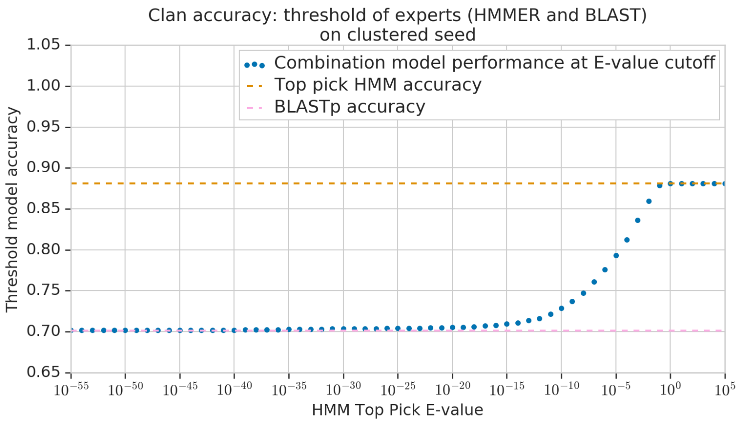


Figure S12: We use the HMMER e-value to combine BLASTp and HMMER predictions on the clustered split. (A) Performance of BLASTp and Top pick HMM on the clustered seed dataset, stratified by the HMMER e-value for each held-out test sequence. (B) Model performance for a combined approach that uses HMM predictions for low HMMER e-values, and BLASTp predictions for high HMMER e-values. We note that combining these two models based on e-value does not create a model that is better than the Top pick HMM model alone.

	Number of examples	Number of families
Train	1086741	17929
Dev	126171	13071
Test	126171	13071

Table S1: **Number of examples for the randomly split Pfam-seed data.**

Sequence Identity Interval	Number of Sequences
10-20	292
20-30	3628
30-40	9537
40-50	16798
50-60	22662
60-70	28277
70-80	40221
80-90	4429
90-100	256

Table S2: Number of sequences per sequence identity bucket for the random Pfam-seed split.

Sequence Identity Interval	Number of Sequences
12-16	33
16-20	259
20-24	849
24-28	1614
28-32	2499
32-36	3569
36-40	4634

Table S3: Number of sequences per sequence identity bucket for more remote sequences of random Pfam-seed split in Figure S3A.

	Number of examples	Number of families
Train	1296280	17929
Dev	21510	4323
Test	21293	3097

Table S4: **Number of examples for the clustered split of the Pfam-seed data.**

Sequence Identity Interval	Number of Sequences
10-12	62
12-14	426
14-16	1058
16-18	2516
18-20	4419
20-22	6013
22-24	4892
24-25	1902

Table S5: Number of sequences per sequence identity bucket for clustered Pfam-seed split.

Model	Error rate	Number of errors
Top Pick HMM	15%	3552
phmmer	31%	6453
BLASTp	34%	7090
ProtCNN	25%	5117
ProtENN	**10%**	**2174**

Table S6: **Alternative approach to calculating performance on the clustered split data.** In both [9] and our work, clustering is used to split the data into train and test sets, however, our construction of a test set is slightly different than that of [9]. We first choose which clusters will be in the test set, and then include all sequences belonging to these clusters in the set. In [9], a single sequence is used for each cluster, since this helps ensure that clusters with many elements do not dominate the accuracy calculation. We instead report the expected value of this randomized procedure by first computing per-cluster average performance and then averaging these to obtain dataset-level performance. In practice, the difference between the evaluation approach in this table and in the main paper is minor because many clusters in our test set are singletons.

Prediction Method	Overall Error Rate	Small Family Error Rate	Large Family Error Rate
ProtCNN	0.495%	3.380%	0.479%
ProtREP (Per-Family)	0.653%	0.651%	0.986%
ProtREP (Per-Instance)	0.510%	0.502%	1.972%

Table S7: Performance when classifying using nearest neighbors in embedding space on the random split.

Prediction Method	Overall Error Rate	Small Family Error Rate	Large Family Error Rate
ProtCNN	27.624 %	82.126 %	25.987%
ProtREP (Per-Family)	18.574 %	17.957 %	39.130%
ProtREP (Per-Instance)	24.560 %	23.704 %	54.428%

Table S8: Performance when classifying using nearest neighbors in embedding space on the clustered split.

Prediction Method	# Founder Sequences per Small Family	Overall Error Rate	Small Family Error Rate
ProtCNN	0	0.427%	100.0%
ProtREP	1	0.790%	14.9%
ProtREP	2	0.764%	9.0%
ProtREP	all available	0.741%	0.7%
Top Pick HMM	1	1.432%	9.3%
Top Pick HMM	all available	1.414%	1.1%

Table S9: **Random split performance at annotating unseen small families of ProtCNN, TPHMM and ProtREP, which uses the learned representation of sequence space.** Small families are defined as those 5568 families that each have <10 train sequences in the random split. The held-out test set contains 710 sequences from these families (see methods), which are used to compute the error rate. ProtREP accuracy at classifying test sequences from unseen small families imrpoves rapidly as founder sequences are provided.

Sequence Identity Interval	Number of Sequences
20-30	12135
30-40	78075
40-50	182431
50-60	331440
60-70	506826
70-80	720693
80-90	1068624
90-100	1656757

Table S10: Number of sequences per sequence identity bucket for random Pfam-full split.

Sequence Identity Interval	Number of Sequences
20-22	137
22-24	629
24-26	1792
26-28	3600
28-30	5977
30-32	8967
32-34	12090
34-36	15311
36-38	19530
38-40	22177

Table S11: Number of sequences per sequence identity bucket for more remote sequences of random Pfam-full split.

Table S12				
Sequence	**ProtENN_call_range**	**Accession**	**Description**	**Comment**
P69905 human hemoglobin	(6, 106)	PF00042.22	Globin	
	(6, 107)	PF00042.22	Globin	
	(7, 105)	PF00042.22	Globin	
	(7, 106)	PF00042.22	Globin	
	(7, 107)	PF00042.22	Globin	This is exactly the HMMER call
	(7, 108)	PF00042.22	Globin	
	(7, 109)	PF00042.22	Globin	
	(7, 110)	PF00042.22	Globin	
	(7, 111)	PF00042.22	Globin	
	(7, 112)	PF00042.22	Globin	
	(7, 113)	PF00042.22	Globin	
	(8, 106)	PF00042.22	Globin	
	(8, 107)	PF00042.22	Globin	
	(8, 108)	PF00042.22	Globin	
	(8, 109)	PF00042.22	Globin	
	(8, 110)	PF00042.22	Globin	
	(9, 107)	PF00042.22	Globin	
	(10, 107)	PF00042.22	Globin	
Q8X7B7 E. Coli TrpCF	(2, 253)	PF00218.21	Indole-3-glycerol phosphate synthase	
	(3, 253)	PF00218.21	Indole-3-glycerol phosphate synthase	
	(5, 253)	PF00218.21	Indole-3-glycerol phosphate synthase	
	(6, 253)	PF00218.21	Indole-3-glycerol phosphate synthase	This is exactly the HMMER call
	(6, 254)	PF00218.21	Indole-3-glycerol phosphate synthase	
	(6, 255)	PF00218.21	Indole-3-glycerol phosphate synthase	
	(7, 253)	PF00218.21	Indole-3-glycerol phosphate synthase	
	(258, 449)	PF00697.22	N-(5'phosphoribosyl)anthranilate (PRA) isomerase	This is exactly the HMMER call
P01308 human insulin	(28, 108)	PF00049.18	Insulin/IGF/Relaxin family	
	(28, 109)	PF00049.18	Insulin/IGF/Relaxin family	This is exactly the HMMER call
	(29, 109)	PF00049.18	Insulin/IGF/Relaxin family	
O00180 human potassium channel	(76, 158)	PF07885.16	Ion channel	
	(76, 159)	PF07885.16	Ion channel	
	(77, 157)	PF07885.16	Ion channel	
	(77, 158)	PF07885.16	Ion channel	
	(77, 159)	PF07885.16	Ion channel	
	(78, 156)	PF07885.16	Ion channel	
	(78, 157)	PF07885.16	Ion channel	
	(78, 158)	PF07885.16	Ion channel	
	(78, 159)	PF07885.16	Ion channel	
	(79, 155)	PF07885.16	Ion channel	
	(79, 156)	PF07885.16	Ion channel	
	(79, 157)	PF07885.16	Ion channel	
	(79, 158)	PF07885.16	Ion channel	
	(79, 159)	PF07885.16	Ion channel	
	(80, 156)	PF07885.16	Ion channel	
	(80, 157)	PF07885.16	Ion channel	
	(80, 158)	PF07885.16	Ion channel	
	(80, 159)	PF07885.16	Ion channel	
	(81, 155)	PF07885.16	Ion channel	
	(81, 156)	PF07885.16	Ion channel	
	(81, 157)	PF07885.16	Ion channel	
	(81, 158)	PF07885.16	Ion channel	
	(81, 159)	PF07885.16	Ion channel	
	(82, 155)	PF07885.16	Ion channel	
	(82, 156)	PF07885.16	Ion channel	
	(82, 157)	PF07885.16	Ion channel	

517

Table S12				
Sequence	ProtENN_call_range	Accession	Description	Comment
	(82, 158)	PF07885.16	Ion channel	This is exactly the HMMER call
	(82, 159)	PF07885.16	Ion channel	
	(83, 157)	PF07885.16	Ion channel	
	(83, 158)	PF07885.16	Ion channel	
	(83, 159)	PF07885.16	Ion channel	
	(85, 158)	PF07885.16	Ion channel	
	(87, 157)	PF07885.16	Ion channel	
	(87, 158)	PF07885.16	Ion channel	
	(87, 159)	PF07885.16	Ion channel	
	(88, 157)	PF07885.16	Ion channel	
	(88, 158)	PF07885.16	Ion channel	
	(189, 265)	PF07885.16	Ion channel	
	(189, 266)	PF07885.16	Ion channel	
	(189, 267)	PF07885.16	Ion channel	
	(189, 268)	PF07885.16	Ion channel	
	(189, 269)	PF07885.16	Ion channel	
	(189, 270)	PF07885.16	Ion channel	
	(189, 271)	PF07885.16	Ion channel	
	(189, 272)	PF07885.16	Ion channel	
	(189, 273)	PF07885.16	Ion channel	
	(190, 263)	PF07885.16	Ion channel	
	(190, 264)	PF07885.16	Ion channel	
	(190, 265)	PF07885.16	Ion channel	
	(190, 266)	PF07885.16	Ion channel	
	(190, 267)	PF07885.16	Ion channel	
	(190, 268)	PF07885.16	Ion channel	
	(190, 269)	PF07885.16	Ion channel	
	(190, 270)	PF07885.16	Ion channel	
	(190, 271)	PF07885.16	Ion channel	
	(190, 272)	PF07885.16	Ion channel	
	(190, 273)	PF07885.16	Ion channel	
	(190, 274)	PF07885.16	Ion channel	
	(191, 265)	PF07885.16	Ion channel	
	(191, 266)	PF07885.16	Ion channel	
	(191, 267)	PF07885.16	Ion channel	
	(191, 268)	PF07885.16	Ion channel	
	(191, 269)	PF07885.16	Ion channel	
	(191, 270)	PF07885.16	Ion channel	
	(191, 271)	PF07885.16	Ion channel	This is exactly the HMMER call
	(191, 272)	PF07885.16	Ion channel	
	(191, 273)	PF07885.16	Ion channel	
	(191, 274)	PF07885.16	Ion channel	
	(192, 265)	PF07885.16	Ion channel	
	(192, 266)	PF07885.16	Ion channel	
	(192, 267)	PF07885.16	Ion channel	
	(192, 268)	PF07885.16	Ion channel	
	(192, 269)	PF07885.16	Ion channel	
	(192, 270)	PF07885.16	Ion channel	
	(192, 271)	PF07885.16	Ion channel	
	(192, 272)	PF07885.16	Ion channel	
	(192, 273)	PF07885.16	Ion channel	
	(192, 274)	PF07885.16	Ion channel	
	(193, 265)	PF07885.16	Ion channel	
	(193, 266)	PF07885.16	Ion channel	
	(193, 267)	PF07885.16	Ion channel	
	(193, 269)	PF07885.16	Ion channel	

Table S12				
Sequence	**ProtENN_call_range**	**Accession**	**Description**	**Comment**
	(193, 270)	PF07885.16	Ion channel	
	(193, 271)	PF07885.16	Ion channel	
	(193, 272)	PF07885.16	Ion channel	
	(193, 273)	PF07885.16	Ion channel	
	(194, 270)	PF07885.16	Ion channel	
	(196, 270)	PF07885.16	Ion channel	
	(197, 269)	PF07885.16	Ion channel	
	(197, 270)	PF07885.16	Ion channel	
	(197, 272)	PF07885.16	Ion channel	
	(197, 273)	PF07885.16	Ion channel	
	(198, 270)	PF07885.16	Ion channel	
Q46899 E. Coli CRISPR cascade	(5, 346)	PF09344.10	CT1975-like protein	
	(5, 347)	PF09344.10	CT1975-like protein	
	(5, 352)	PF09344.10	CT1975-like protein	
	(5, 353)	PF09344.10	CT1975-like protein	
	(5, 354)	PF09344.10	CT1975-like protein	
	(5, 355)	PF09344.10	CT1975-like protein	
	(5, 356)	PF09344.10	CT1975-like protein	
	(5, 357)	PF09344.10	CT1975-like protein	
	(5, 358)	PF09344.10	CT1975-like protein	This is exactly the HMMER call
	(5, 359)	PF09344.10	CT1975-like protein	
	(5, 360)	PF09344.10	CT1975-like protein	
Q5BJI7 Zebrafish methyltransferas	(18, 241)	PF00856.28	SET domain	This is exactly the HMMER call. There is a missing call for a zinc finger, but the true call for this domain has length < 50, which is by design excluded from these calls.
Q8R5M8 Mouse Ig	(216, 318)	PF13205.6	Bacterial Ig-like domain	This is a false positive, and is a match to E-set, not Ig clan. E-set is an immunoglobulin-like fold, so it's not a particularly terrible false positive. Notably, there are also two false negatives on this protein, beacuse there are a missing Ig call for the ranges 52-146 and 151-233
	(237, 320)	PF13927.6	Immunoglobulin domain	
	(239, 320)	PF13927.6	Immunoglobulin domain	
	(241, 320)	PF13927.6	Immunoglobulin domain	
	(243, 320)	PF13927.6	Immunoglobulin domain	
	(244, 320)	PF13927.6	Immunoglobulin domain	
	(245, 319)	PF13927.6	Immunoglobulin domain	
	(245, 320)	PF13927.6	Immunoglobulin domain	This is exactly a HMMER call
	(245, 321)	PF13927.6	Immunoglobulin domain	
	(246, 320)	PF13927.6	Immunoglobulin domain	
	(247, 320)	PF13927.6	Immunoglobulin domain	
	(248, 320)	PF13927.6	Immunoglobulin domain	
	(249, 320)	PF13927.6	Immunoglobulin domain	
	(250, 320)	PF13927.6	Immunoglobulin domain	
	(251, 320)	PF13927.6	Immunoglobulin domain	
	(252, 320)	PF13927.6	Immunoglobulin domain	
	(254, 320)	PF13927.6	Immunoglobulin domain	
	(255, 320)	PF13927.6	Immunoglobulin domain	
	(256, 320)	PF13927.6	Immunoglobulin domain	
	(257, 320)	PF13927.6	Immunoglobulin domain	

Table S12				
Sequence	**ProtENN_call_range**	**Accession**	**Description**	**Comment**
	(309, 373)	PF13290.6	Chitobiase/beta-hexosaminidase C-terminal domain	This is a false positive for an E-set, which again is not too bad of a false positive since E-set is an Ig-like fold
S5S176 yeast dehydrogenase	(29, 140)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(30, 138)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(30, 139)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(30, 140)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(30, 141)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(30, 142)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 136)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 137)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 138)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 139)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 140)	PF08240.12	Alcohol dehydrogenase GroES-like domain	This is exactly the HMMER call
	(31, 141)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 142)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 143)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 144)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 145)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(31, 146)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 136)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 137)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 138)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 139)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 140)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 141)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 142)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 143)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 144)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 145)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 146)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(32, 149)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(33, 138)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(33, 139)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(33, 140)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(33, 141)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(33, 142)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(33, 143)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(33, 144)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(33, 149)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(34, 140)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(34, 141)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(34, 142)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(34, 143)	PF08240.12	Alcohol dehydrogenase GroES-like domain	
	(181, 311)	PF00107.26	Zinc-binding dehydrogenase	
	(182, 307)	PF00107.26	Zinc-binding dehydrogenase	
	(182, 308)	PF00107.26	Zinc-binding dehydrogenase	
	(182, 309)	PF00107.26	Zinc-binding dehydrogenase	
	(182, 310)	PF00107.26	Zinc-binding dehydrogenase	
	(182, 311)	PF00107.26	Zinc-binding dehydrogenase	This is exactly the HMMER call
	(182, 312)	PF00107.26	Zinc-binding dehydrogenase	
	(182, 314)	PF00107.26	Zinc-binding dehydrogenase	
	(182, 315)	PF00107.26	Zinc-binding dehydrogenase	
	(183, 311)	PF00107.26	Zinc-binding dehydrogenase	
Q505I0 frog kinase	(126, 423)	PF00454.27	Phosphatidylinositol 3- and 4-kinase	This is exactly the HMMER call

Table S12				
Sequence	**ProtENN_call_range**	**Accession**	**Description**	**Comment**
Q17766 nematode MFS	(28, 89)	PF04117.12	Mpv17 / PMP22 family	This call is a false positive. There is also a false negative (missing call) to Folate Carrier from 1-403.

Table S12: **ProtENN predicted domain boundaries sequence closely match HMMER output for a diverse set of 10 proteins.** Predicted domain boundaries are computed by sliding the ProtENN classifier over all `start` − `end` ranges, to identify the set of ranges where the confidence is highest (equal to 1). We find that confident ranges very closely match the domain boundaries computed by HMMER for most proteins. We only include predicted domains longer than 50 residues for ProtENN calls, because shorter ranges lead to spurious calls, as is seen often with HMMER, especially with regions that include repeats [21].

Model Type	Hyperparameter	Search Range
ProtCNN	batch size	32, 64, 128, 256
	dilation rate	1, 2, 3, 5
	filters	300 thru 3000, increments of 100
	ResNet block of first dilated layer	2, 3
	kernel size	3, 7, 9, 11, 21, 31
	ResNet layers	1 thru 6
	learning rate	1e-05, 5e-05, 1e-4, 5e-4, 1e-3
	learning rate decay steps	1e3,1e4,1e6, decay off
	pooling	max, mean
kmer	embedding rank	100, 1000, 10000
	learning rate	1e-4, 5e-4, 1e-3
	ngram order	1 thru 5

Table S13: Search ranges for hyperparameter values for ProtCNN.

Parameter	Value
batch size	32
dilation rate	3*
filters	1100*
first dilated layer	2*
gradient clip	1
kernel size	9*
learning rate	.0001*
learning rate decay rate	0.997
learning rate decay steps	1000*
learning rate warmup steps	3000
number of ResNet layers	5*
pooling	max*
ResNet bottleneck factor	0.5
train steps	500000**

Table S14: Hyperparameters used in ProtCNN with the Pfam-seed dataset. An asterisk denotes a tuned value. Two asterisks denote that the model was overfit, and the number of tuning steps was chosen post-hoc so as to maximize dev-set performance.

Model Type	Search Algorithm	Approx. number of samples
CNN (all depths)	random sampling	17000 *
kmer	random sampling	50

Table S15: Search algorithms and number of samples for hyperparameter tuning, by model. The asterisk denotes that many of these configurations were not feasible, as they did not fit in GPU memory.

Parameter	Value
batch size	64
filters	2000*
first dilated layer	NA
gradient clip	1
kernel size	21*
learning rate	.001*
learning rate decay rate	0.997
learning rate decay steps	1000*
learning rate warmup steps	3000
pooling	max*
ResNet bottleneck factor	0.5
train steps	1100000**

Table S16: Hyperparameters used in ProtCNN with the Pfam-full dataset. An asterisk denotes a tuned value. Two asterisks denote that the model didn't necessarily converge, but was ended after a reasonable time training (17 days).

Program	Average inference speed (sequences per core-second)	Estimated runtime on Pfam-seed test (hrs)
hmmsearch	24.4	1.4
hmmscan	2.2	16.2
BLASTp	1.1	30.5
ProtCNN (CPU only)	9.7	3.6
ProtCNN (GPU)	376.6	0.1

Table S17: Inference speed of `hmmscan`, `hmmsearch`, and `blastp` run on sandboxed n1-standard-32 (32-core, 120 GB RAM) Google Cloud Platform instances with all data in main memory and using a single core. The ProtCNN model was run in a similar configuration on a n1-standard-8 instance (8-core, 32 Gb RAM) using a single CPU thread for ProtCNN (CPU only), and additionally, one NVIDIA P100 GPU accelerator for ProtCNN (GPU). Additional details, including commands used, are available in the supplement.

	kmer
batch size	64
gradient clip	1
learning rate	.0005*
learning rate decay rate	0.997
learning rate decay step	1000
learning rate warmup steps	3000
kmer order	2*
number of hash buckets	10000*
train steps	300000

Table S18: Hyperparameters used in kmer benchmark models. An asterisk denotes a tuned value.

Sequence Name	Residues	Sequence
AT1A1_PIG	161-352	NMVPQQALVIRNGEKMSINAEEVVVG DLVEVKGGDRIPADLRIISANGCKVD NSSLTGESEPQTRSPDFTNENPLETR NIAFFSTNCVEGTARGIVVYTGDRTV MGRIATLASGLEGGQTPIAAEIEHFI HIITGVAVFLGVSFFILSLILEYTWL EAVIFLIGIIVANVPEGLLATVTVCL TLTAKRMARK
V2R_HUMAN	54-325	SNGLVLAALARRGRRGHWAPIHVFIG HLCLADLAVALFQVLPQLAWKATDRF RGPDALCRAVKYLQMVGMYASSYMIL AMTLDRHRAICRPMLAYRHGSGAHWN RPVLVAWAFSLLLSLPQLFIFAQRNV EGGSGVTDCWACFAEPWGRRTYVTWI ALMVFVAPTLGIAACQVLIFREIHAS LVPGPSERPGGRRRGRRTGSPGEGAH VSAAVAKTVRMTLVIVVVYVLCWAPF FLVQLWAAWDPEAPLEGAPFVLLMLL ASLNSCTNPWIY

Table S19: Wildtype sequences, keyed by Uniprot ID, that were used for saturation mutagenesis predictions.