Journal Title, 2024, pp. 1–9

doi:
Paper

PAPER

NEAR: Neural Embeddings for Amino acid Relationships

Daniel R. Olson,[1,*,†] Daphne Demekas,[2,†] Thomas Colligan[2] and Travis J. Wheeler[1,2]

[1]Department of Computer Science, University of Montana, Montana, USA and [2]R. Ken Coit College of Pharmacy, University of Arizona, Arizona, USA
[*]Travis J. Wheeler. twheeler@arizona.edu
[†]These authors contributed equally to this work

Abstract

We present NEAR, a method based on representation learning that is designed to rapidly identify good sequence alignment candidates from a large protein database. NEAR's neural embedding model computes per-residue embeddings for target and query protein sequences, and identifies alignment candidates with a pipeline consisting of k-NN search, filtration, and neighbor aggregation. NEAR's ResNet embedding model is trained using an N-pairs loss function guided by sequence alignments generated by the widely used `HMMER3` tool. Benchmarking results reveal improved performance relative to state-of-the-art neural embedding models specifically developed for protein sequences, as well as enhanced speed relative to the alignment-based filtering strategy used in `HMMER3`'s sensitive alignment pipeline.

Key words: Neural embedding, Sequence annotation

Introduction

The ease and low cost of DNA sequencing creates unprecedented opportunity to catalog and understand the genomic diversity of life on Earth. One impact of the ongoing data deluge is that computational tools for sequence annotation should be designed with massive-scale search in mind [3, 31, 12]. Another is that new sequence data sets often include proteins that defy current annotation efforts – they are either entirely novel, or have diverged so far from their ancestral sequence that current methods are unable to detect their homology to already known sequence. The challenge is particularly great when annotating metagenomic datasets as they often reach terabases in size and large fractions (and in many cases, the majority) of putative proteins may go unannotated [6, 32, 36] due to a combination of novelty, diversity, and sequencing error.

To meet these challenges, researchers continue to make advances to annotation methods along classical avenues of algorithm development for speed [46, 4, 43] and statistical model design for improved sensitivity and error handling [9, 29, 15]. Meanwhile, new general natural language processing (NLP) strategies for annotation have gained traction, fueled by representation learning with neural networks [1]. NLP representation strategies have advanced rapidly from word2vec style [35] representation, to attention-based approaches [48] including BERT (Bidirectional Encoder Representations from Transformers [2]), T5 (Text-to-Text Transfer Transformer [41]), and GPT (Generative Pretrained Transformer [40]). Biosequence analogs to these methods abound, such as [27, 2, 13, 42], though these models are not necessarily suited to sequence annotation.

In the neural representation framework for biosequences, a neural network computes a vector representation for a sequence, so that each sequence is embedded in a high dimensional representation space. A well-behaving network will embed a pair of sequences near each other in space if they share similar properties and place dissimilar pairs far apart.

Steinneger et al. [44] have shown that neural representation methods appear to identify some homologs of query proteins that are not found using sequence alignment methods. Importantly, the authors also showed that vector distances computed using their best-performing model (knnProtT5) struggled to distinguish decoys from true positives, so that the model was more useful when its top matches were re-scored by computing sequence alignments – in other words, it appears to be preferable to treat knnProtT5 as a filter for more expensive sequence alignment. (Note: even under this re-scoring strategy, the knnProtT5-based approach showed a loss in overall recall relative to MMseqs2).

In this work, we embrace and extend the idea of using neural embeddings as the basis of a pre-filter for alignment-based annotation. We introduce a network and training method that computes a context-informed amino acid embedding such that two residues will be close in embedding space if they are likely to be placed in the same column of a pairwise sequence alignment. To find candidate homologs for a query protein sequence q, our pre-filter implementation (NEAR – Neural Embeddings for

© The Author 2024. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

Amino acid Relationships) computes an embedding for each residue in q, searches a target dataset of residue embeddings T for near neighbors of each of those query residue embeddings, then extracts sequences from T that share many near neighbors with q.

For a filter to be effective in a pipeline that seeks to find homologs of a query sequence q in a target set T or candidate proteins, it must identify a small subset of T that is highly enriched for the desired true positives, and it must do so quickly. We discuss each of these issues below.

Filtering with high sensitivity

Because the most valuable filter is one that will be effective for maximally-sensitive downstream processing, we approach methods development and evaluation in the context of state of art annotation with profile hidden Markov models (pHMMs [30, 8, 9]). Profile HMMs show greater sensitivity [25, 29, 46] than other homology search methods such as BLAST [5], LAST [26], and MMseqs2 [46]. The sensitivity of pHMMs is due to a combination of (i) position specific scores [19] learned from sequence family members and (ii) implementation of the Forward algorithm [39, 30], which sums the probabilities of all possible alignments between the aligned pair of sequences. The Forward algorithm is responsible for much of the sensitivity gains of pHMMs, but is computationally expensive. HMMER3 [10] introduced a pipeline in which most candidates are never subjected to the most computationally expensive analysis, thanks in large part to a stage called MSV that performs highly-optimized ungapped sequence alignment to identify promising seeds. By default, HMMER3's MSV stage filters away all but ∼2% of decoy (non-homologous) sequences.

The role of MSV in the HMMER3 pipeline is the role that our neural embedding model, NEAR, is designed to fill: given a large set T of target protein sequences and a query protein sequence, rapidly identify a small subset of target sequences that are expected to produce high Forward scores, so that a large majority of the unrelated sequences in T can be ignored (filtered) without further consideration.

Filtering for speed

Ideally, a neural filter such as NEAR will filter as effectively as HMMER3's MSV filter with greater speed. To achieve acceptable speed, a search method must store the set of target embeddings in an efficient data structure that supports rapid identification of k nearest residue neighbors without computing distances for all neighbors. Fast approximate near neighbor search in high volume and high dimensional data is a highly researched area [7, 17, 20, 34, 22], so options for achieving fast neighbor search are plentiful. NEAR uses the Facebook AI Similarity Search (FAISS) library [24].

Manuscript focus and software availability

In the sections below, we provide a description of methods for model design, training, and validation, then demonstrate that NEAR's similarity calculations enable recovery of true matches with greater filtering efficacy than other neural embedding strategies and can be produced ∼10x faster than the estimates of HMMER's MSV filter. NEAR is available under an open license at `https://github.com/TravisWheelerLab/NEAR`.

Methods

NEAR is designed to serve as a fast and sensitive pre-filter for large-scale sequence homology search tools. Given a set of query sequences and a set of target sequences, NEAR filters down the set of all query-target pairs to a much smaller set of pairs that are good candidates for sequence alignment in a later analysis step. NEAR accomplishes its goal with two steps. First, NEAR employs a neural network to represent sequences of residues as sequences of high-dimensional vectors, each vector describing a specific residue and the sequence near that residue, i.e its *context*. Next, NEAR uses the FAISS library to compute an efficient search index for the target vectors, and then to search query embeddings against the target index.

Training

NEAR's neural network (described in Model architecture) transforms a sequence of amino acids, $S = \{S_1, S_2, ..., S_n\}$ into a sequence of high dimensional vectors, $\{\vec{s_1}, \vec{s_2}, ..., \vec{s_n}\}$. Each vector $\vec{s_i}$ represents the corresponding residue, S_i, in a manner that reflects the residue's surrounding context. NEAR learns to embed sequences so that residues that locally align to one another will be embedded as vectors that have a large dot product. For example, consider two homologous sequences A and B. NEAR is trained so that $\vec{a_i} \cdot \vec{b_j}$ will be large when A_i aligns with B_j, even when $A_i \neq B_j$. NEAR is trained to look at the context around a residue, and create an embedding that represents how that residue and context are expected to locally align with other residues and contexts.

As shown in Figure 1, two homologous sequences, A and B, are aligned using HMMER. The alignment is then used to produce a target matrix, T, where $T_{ij} = 1$ when A_i aligns with B_j, and is 0 everywhere else. NEAR's neural network is then used to embed A and B into vector sequences $\{\vec{a_i}\}$ and $\{\vec{b_j}\}$. Importantly, A and B are embedded independently of one another; the neural network does not use the embedding of one sequence to inform its embedding of the other sequence. The embeddings are used to create a inner product matrix, D, where $D_{ij} = \vec{a_i} \cdot \vec{b_j}$.

N-Pair loss [45], is used to drive $\vec{a_i}$ and $\vec{b_j}$ in the same direction if their residues align with each other, and to drive their direction away from each other if they don't align with each other. We also add an $L2$ regularization term to our loss (weighted by a scalar, γ) which discourages embeddings from being unnecessarily large

$$L = \underbrace{\sum_{i,j}^{n}(-T_{ij}\log(\overbrace{\frac{e^{D_{ij}}}{\sum_{w}^{n} e^{D_{iw}}}}^{\textbf{Softmax}}))}_{\textbf{N-Pair Loss}} + \underbrace{\frac{\gamma}{N}(\sum_{i}^{n} a_i^2 + \sum_{j}^{n} b_j^2)}_{\textbf{L2 Regularizaiton}} \quad (1)$$

Note that the loss is only applied to aligned vector pairs, i.e i, j pairs where $T_{ij} = 1$. Importantly, vector pairs for unaligned residues will still propagate meaningful gradient (Figure 2) because the loss function does not use D directly, but instead uses softmax(D). When $T_{ij} = 1$ the loss is improved by both increasing the softmax numerator, $e^{D_{ij}} = e^{\vec{a_i} \cdot \vec{b_i}}$, and also by decreasing the denominator, $\sum_{w}^{n} e^{D_{iw}}$. In this formulation, high similarity between unaligned vectors $\vec{a_i}$ and $\vec{b_w}$ will increase the denominator, causing a reduction in the softmax value of the aligned vectors, $\vec{a_i}$ $\vec{b_j}$. The result will be a loss value that is large when unaligned residues share similar embeddings.

NEAR was optimized using ADAM [28] with a learning rate of 1e-5 and a batch size of 64.

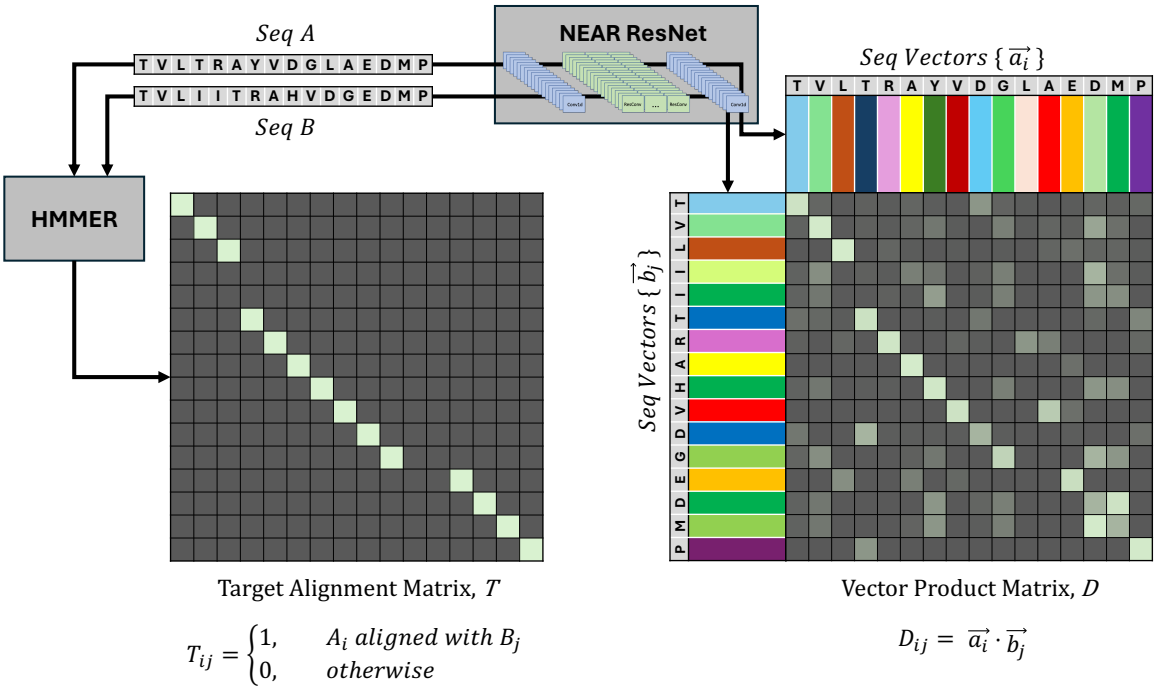

Fig. 1. During training, two homologous sequences, A and B, are aligned with HMMER to create a target matrix T. The sequences are also passed through NEAR's neural network to produce sequences of vectors, $\{\vec{a_i}\}$ and $\{\vec{b_j}\}$

. We calculate the dot product between every $\vec{a_i}$and $\vec{b_j}$ pair, and store the result in a matrix, D.

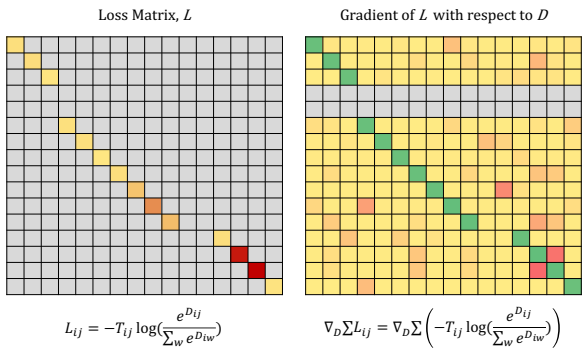

Fig. 2. The loss signal produced by applying the loss function in Equation 1 to T and D in 1 (left), and the loss gradient with respect to D (right). When $\vec{a_i}$ and $\vec{b_j}$ are aligned to one another, the gradient tries to increase the softmax numerator by pushing the vectors closer together. When they $\vec{a_i}$ and $\vec{b_j}$ not aligned, the gradient tries to decrease the softmax denominator by pulling the vectors further apart.

Model architecture

NEAR is implemented as a 1D Residual Convolutional Neural Network [21, 49]. A batch of sequences is initially embedded as a [batch$\times$256$\times$seq length] tensor using a context-unaware residue embedding layer. The tensor is then passed through 8 residual blocks. Each residual block performs two 1D convolutions before adding the result back to the residual block's input via skip-connection. Both convolutions inside a residual block are ELU activated, have a kernel size of 3, and have a filter size of 256. After the residual blocks, the final layer performs a linear projection of the embeddings using a 1D convolution with no activation function, a kernel size of 1, and a filter size of 256. NEAR was implemented with PyTorch [37] and PyTorch Lightning [14].

Sequence selection and processing

Protein sequences used to train and evaluate NEAR were sampled from UniRef90 [47], which consists of protein sequences that are no more than 90% identical to each other. 1,000,000 sequences were randomly selected from UniRef90, then split to target (900,000) and query (100,000) subset.

The 900,000 target sequences were further split into two clusters for training and validation/testing: $T1$ ($\sim$80%) and $T2$ ($\sim$20%). If sequences in the test set are close homologs to sequences in the training set, a model may be rewarded for effectively memorizing its training data. To avoid this risk, we clustered target sequences at 30% identity using UCLUST [11], then assigned sequences to subsets $T1$ and $T2$ such that sequences sharing a cluster were placed collectively in one or the other. The resulting target sets were of size 720,005 ($T1$) and 179,995 ($T2$) sequences. Query sequences were split into three subsets: Q_{train} (64,000 sequences), Q_{validate} (16,000 sequences), and Q_{test} (20,000 sequences),

For training, `phmmer` from HMMER3 [10] was used with the '–max' flag enabled to identify the set of alignments that an ideal filter might hope to return. Q_{train} was searched against $T1$ (64,000 vs 720,005 sequences), resulting in 38,582,011 training alignments with E-value ≤ 1. To embed the sequences making up an alignment, the sequences were extracted independently from the alignment and trimmed to length of 128. Sequences shorter than 128 residues were lengthened by sampling random amino acids according to the distribution of aminos found

in biological sequences. For computing the loss function, all alignments were length-capped by considering only the first 128 alignment columns. (Note: this generally means that the last few residues of the embedded sequences are not represented in the alignment; these embeddings are not included in the loss function computation, as they do not represent aligned amino acids.)

Validation data for hyperparameter tuning consisted of 5,202,590 alignments with E-value ≤ 10 produced with `phmmer --max` comparison between Q_{validate} and $T2$ (16,000 vs 179,995 sequences); alignments were trimmed to length 128.

For testing, the 20,000 held-out query sequences in Q_{test} were filtered to remove sequences longer than 512 residues, leaving a set of 16,768 test queries called Q_{test512}. The 512-residue length limit was imposed due to memory limitations for one of the alternative models (ProteinBERT [2]). Sequences in the evaluation set $T2$ were similarly length-filtered, resulting in 139,900 target sequences with length $\leq$512, called $T2_{512}$. These two sets were used to compute both ground truth alignments (treating HMMER alignments as the target truth) and all evaluated filter variants. Note that NEAR was trained on 128-length alignments, but testing is performed on alignments of sequences of all lengths up to 512, demonstrating that NEAR's embedding model generalizes to sequences that are longer than the sequences and alignments that it was trained on.

Hyperparameter tuning

To optimize model performance, we conducted hyperparameter tuning, specifically adjusting the learning rate, number of filters, kernel size, and the number of residual blocks. We systematically explored three different values for each parameter, resulting in a total of 81 training runs. The final set of hyperparameters was selected based on their ability to minimize the validation loss on the holdout ($\sim$15%) validation set.

Indexing and searching embeddings with FAISS

NEAR initiates search by computing residue embeddings for a set of target proteins. These embeddings are used to generate a search index with the FAISS library [24] for efficient similarity search in high dimensions. When running in CPU-only mode, NEAR uses the `IndexIVF` FAISS index, with no quantization. When running with a GPU, NEAR uses the `IndexIVFPQ32` index, which uses product quantization [23] to improve search speed. Both `IndexIVF` and `IndexIVFPQ32` work by partitioning target embeddings into different lists (called *cells*); NEAR parameterizes FAISS to use 5,000 cells by default. After building the index space, FAISS maintains a centroid (mean vector) for each cell, and each embedded residue is assigned to to the cell with the closest centroid.

With the search index created, NEAR computes residue embeddings for query proteins. For each query embedding, cells in the FAISS target index are ranked according to proximity of the cell's centroid to the query embedding. For each of the *nprobe* closest cells, the target embeddings within the cells are evaluated by their distance to the query embedding. The parameter *nprobe* can be increased to improve sensitivity, or decreased to improve search-time. The final result of the FAISS search process is 1,000 approximately nearest neighbors (i.e hits) for each individual query embedding, along with the associated cosine similarities for the hits. When a query and target residue produce a hit with high cosine similarity, then the query and target residues are expected to align well with one another.

To estimate the alignment strength between an entire query and target sequence pair, NEAR computes the sum of the cosine similarities for all hits shared between the query sequence and target sequence. Note that, for a given query sequence, scores will be captured for only a small fraction of the full set of target sequences, because most target sequences will have no residues that produce a top-1000 nearest-neighbor to any of the query residues.

When a target sequence contains a region with strong composition bias or highly repetitive sequence, several residues in that region may appear among the top-k nearest neighbors of a query residue. To diminish the effect of repetitive sequence on search, NEAR imposes a constraint on matching residues: for each query residue, a specific target sequence may only contribute one near neighbor. Even with this constraint in place, highly repetitive sequence still tends to produce high scoring matches, just as with sequence alignment methods. For our evaluations, we masked amino acids reported as repetitive by `tantan` [15] in a fashion analogous to soft-masking strategies in sequence annotation [16]. Specifically, we applied `tantan` on our evaluation query and target sequences with default settings, and residues reported to be repetitive were not added to the embedding space. These amino acids were still included in sequences when computing an embedding for neighboring residues.

Benchmark construction

Evaluation of NEAR and alternative methods was performed on a benchmark consisting of both positive controls and negative decoys.

Positives

Because NEAR is envisioned as a pre-filter for a pHMM tool like HMMER, benchmark 'positives' were determined based on `phmmer` alignments. As described in Sequence selection and processing, sequences were sampled from UniRef90, divided into subsets, and filtered to remove sequences with length >512, resulting in 16,768 query sequences in Q_{test512} and 139,900 target sequences in $T2_{512}$. Repetitive sequence in both sets was masked using the tool `tantan` prior to alignment with `phmmer` with the '--max' flag set. All hits with E-value $\leq$ 10 were considered positives – these are the pairings for which an effective filter will produce a high score.

Decoys

The set of decoy sequences consists of a filtered set of reversed protein sequences. Reversed sequences were used because they preserve residue distribution and the existence of local repeats found in true proteins, while disrupting the original sequence's functional properties. However, the results of [18] suggest that reversed sequences be used as decoys only after removing positive matches in the original (un-reversed) sequences, due to a surprising frequency of high scoring approximate palindromes in strings over all alphabets. Let U be the set of 89,867 sequences from $T2_{512}$ that were included in the hit set for at least one query from Q_{test512} with E-value $\leq$ 1. The decoy set D was created by reversing all remaining 50,033 sequences in $Q_{\text{test512}} - U$. The result is a decoy set D consisting of protein sequences with realistic properties of composition bias and repetitiveness, but with no actual homology (because they are reversed sequences), and no elevated risk for false match due

to approximate palindromes (because homologs were removed prior to reversal).

Evaluation

CPU-based evaluation was performed on a system with 2x AMD EPYC 7642 processor with 48 cores @ 2.4GHz and 512GB RAM. Tests were performed on both 1 thread and parallelized across 16 threads, and used the FAISS `IndexIVFFlat` index to report nearest neighbors from among the *nprobe* cells closest to a query embedding, for *nprobe* between 5 and 20.

GPU-based evaluation was performed on a system with an NVIDIA A100 GPU with 80GB RAM, an AMD EPYC 7713 processor with 64 cores @ 2.5GHz, and 512GB RAM. Due to CUDA memory limitations, the GPU-based implementation employed a FAISS product quantizer (`IndexIVFPQ32`) for approximate nearest neighbor search. Because this quantizer provides a rougher approximation, `IndexIVFPQ32`-based evaluations were reported for *nprobe* values of 50 and 150. Auxiliary tests for speed on commodity hardware were performed on a laptop with an NVIDIA GeForce RTX 3080 Ti Laptop GPU with 12GB RAM, an Intel i7-12800H processor with 14 cores @ 4.7GHz, and 32GB RAM.

Results

All analyses were performed using the positive controls and decoy sequences described in Methods (positives are sequence pairings found in alignments produced by `phmmer --max`, at E-value $\leq$ 10; decoys are reversed real protein sequences, where the set of pre-reversal sequences is those in the target set that show no similarity to the query sequences). To place NEAR results in context, we also tested state-of-the-art neural embedding models for protein sequences: ProtTransT5 [44], Facebook's ESM2 [33], and ProteinBERT [2]. ESM2 and ProteinBERT, like NEAR, compute embeddings of each individual amino acid in a protein sequence, whereas ProtTransT5 generates a single embedding for a sequence.

The ProtTransT5 model is released with an implemented search pipeline, also using `FAISS` to identify near neighbors; we used this match-scoring system for ProtTransT5 testing. Both ESM2 and ProteinBERT are purely amino acid embedding models, intended to be fine-tuned for general purposes; to enable their use for the pre-filter task, we applied their embeddings in the same manner as the embeddings of NEAR. For all three (NEAR, ESM2, ProteinBERT), target sequence residue embeddings were computed offline (prior to search) and the resulting vectors were captured in a FAISS index. At search time, query residues were embedded, neighboring residues were searched using FAISS, and matching sequences were scored as described in Methods (summing over near neighbor residue similarities).

Filtering efficacy for neural embedding models and alignment tools

In all filters explored in this analysis, a score is computed for each candidate alignment, and these scores can be ranked. Ideally, the lowest-scoring positive control will produce a score that is higher than that of the highest-scoring decoy, enabling straightforward selection of a filter score threshold. In practice, some decoys produce higher filter scores than some true positives. The result is that threshold selection leads to a tradeoff between high sensitivity (a low threshold allows the slow and accurate post-filter algorithm to re-score a greater number of candidates) and overall tool speed (a high threshold minimizes the amount of work that must be performed by the post-filter algorithm). Different filters present different tradeoffs, so we sought to explore the filtering utility of NEAR scores relative other filters.

Comparison to neural embedding methods

Each of the neural embedding filters produces a list of scored candidates. By sorting filtered candidates by their score, a curve can be plotted, as in Figure 3, that presents recall of true positives (Y-axis) as a function of efficacy in filtering decoys (X-axis). An ideal filter will remove 100% of decoys while producing 100% recall.

Positive controls are not all equivalently simple to distinguish from decoys. For some positives, `phmmer` produces strong E-values (e.g. $\leq 1e-10$), while other positives may yield more marginal E-values. Tools such as HMMER report hits with E-value as high as 10 by default, though more restrictive cutoffs are often used in practice (for example, Pearson [38] suggests that E-values > 0.001 are "scientifically suspect"). It seems reasonable to expect that a filter might more easily separate decoys from highly-significant positives than from positives with marginal score. To explore this effect, Figure 3 presents recall-vs-filtering plots for positives at varying level of E-value cutoff (as computed by `phmmer`): $\leq 1e-10$, $\leq 1e-4$, ≤ 0.1, and ≤ 10. At all levels, NEAR produces recall that is superior to that of other neural embedding methods at any level of filtration.

Comparison to common alignment tools

Figure 3 also includes the recall-filtering tradeoff curve for scores computed in HMMER3's MSV filter [10]. NEAR produces recall competitive with MSV when filtering 99% of decoys. To provide fuller context, Figure 3 also includes the recall/filtering values for MMseqs2 [46] and LAST [26], run with default parameters and E-value cutoff of 10. MMseqs2 can produce its set of candidates with a score, enabling the creation of a similar recall/filtration curve. We were unable to produce a similar ranked seed output from LAST, so Figure 3 shows the total recall as a function of filtration for the final LAST output (note: LAST produces a hit list with few decoys, so it is plotted as a single point).

Comparison of varying nprobe values

Figure 4 presents recall/filtration results across a few values for FAISS's nprobe parameter. These curves show that the large nprobe value enabled by the quantized GPU index aids slightly in recall, and that there is a modest loss in recall when nprobe values are reduced. The plot presents results only for the set of target positives that produce a HMMER E-value ≤ 0.1, but is representative of the relative recall loss for other E-value cutoffs.

Time analysis

An filtering process should be both effective and fast. Table 1 presents the run time of NEAR and other methods by capturing (1) the time required to produce a pre-processed index of the 139,900 (positives) + 50,033 (decoys) = 189,933 total target sequences (applicable for all methods but HMMER's MSV filter) and (2) the time per query required to find and assign scores to neighbors (including embedding time where appropriate).

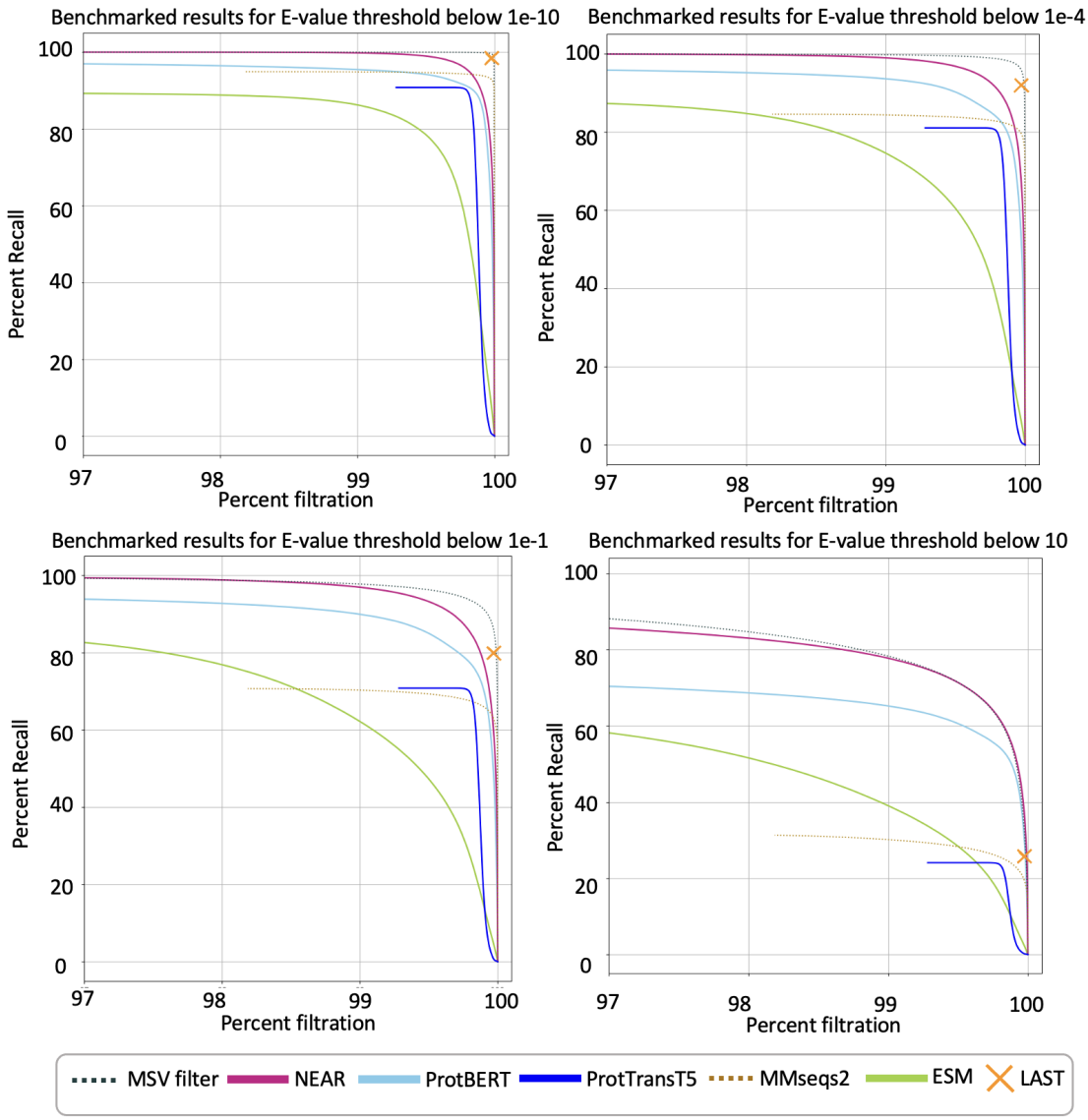

Fig. 3. Recall (Y-axis) as function of filtration rate (X-axis). Each curve was produced by sorting candidate matches by descending score and capturing the level of recall (fraction of positives) and filtration (fraction of removed decoys) across the range of scores. In these plots, an ideal curve remains high and to the right. Recall/filtration curves are plotted for NEAR (GPU, nprobe=150) and other neural embedding methods ProtTransT5, ProteinBERT, and ESM2, as well as for the string-based pre-filters of classical aligners in the form of the HMMER3's MSV filter (default settings) and MMseqs2 pre-filter (-s 7.5 -max-seqs=1000). Embedding-based filters are presented with solid lines, while string-based approaches (MSV and the MMseqs2 pre-filter) are shown with dotted curves. A single point is presented for the alignment software LAST, which produces a final hit list that filters nearly all decoys. Plots are provided for four E-value thresholds used to define the set of positively recalled hit: 1e-10, 1e-4, 0.1, and 10.

Table 1 provides per-query averages of the time taken to run embedding and alignment based filters calculated for evaluating 16,769 queries of maximum length 512 against a target database of 151,287 queries of the same maximum length. Where possible, each tool was evaluated in both GPU and CPU search: on a server-grade GPU (NVIDIA A100) and parallelized across 16 cores. ProteinBERT index construction and search could only be performed on CPU due to a conflict between embedding size and FAISS quantization. ProtTransT5 implementation performed embeddings on GPU, but search on CPU (16 cores). Embedding time is responsible for $< 10\%$ of total time for NEAR models, and close to 30% of time for others. To explore NEAR's utility on commodity hardware, we also tested it on a laptop-based commodity GPU (3080 Ti). Table 1 also presents index construction and search times for HMMER3's MSV filter, along with MMseqs2's pre-filter and LAST's total time, all parallelized across 16 CPUs.

Discussion

Here, we have described NEAR, a neural network search pipeline that produces better recall than state of the art neural embedding models for protein sequences, with speed greater than the optimized filter using in HMMER, MSV. The residue embedding models compared against (ESM and ProteinBERT)

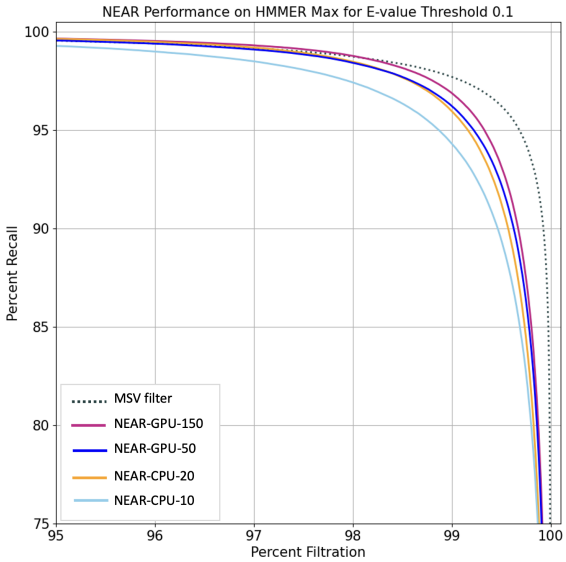

Fig. 4. Recall (Y-axis) as function of filtration rate (X-axis) for nprobe variants of NEAR. Each curve was produced by sorting candidate matches by descending score and capturing the level of recall (fraction of positives) and filtration (fraction of removed decoys) across the range of scores.

Methods	Indexing Time (s)	Search Time (s/query)
HMMER3 MSV filter	-	0.290
NEAR GPU (A100)	67.5	0.037
NEAR GPU (3080 Ti)	81.3	0.059
NEAR CPU (16 cores)	861.2	0.253
ESM GPU (A100)	123.3	0.168
ESM CPU (16 cores)	2661.9	3.430
ProtTransT5*	15.2	0.104
ProteinBERT CPU (16 cores)	2252.4	1.760
MMseqs2 Prefilter	31.0	0.016
LAST	7.0	0.170

Table 1. Index Creation and Search times for various models/search tools, with respect to a target database of 189,933 sequences of maximum length 512 (139,900 real proteins + 50,033 reversed decoys). For neural embedding methods, embedding time is included in time for index construction and neighbor search. ProteinBERT embeddings could not be indexed with the FAISS GPU index due to conflicts between ProteinBERT vector size and FAISS requirements. Indexing and search times for ESM and ProteinBERT exceed times for NEAR due to their larger embedding vector size. LAST runtime is for the full alignment pipeline.
* ProtTransT5 requires that index construction be performed with a GPU, while search is performed with a CPU.

are both transformer models, and produce embeddings with dimensionality ∼4x the size of NEAR's embeddings – the models are therefore bigger (thus perhaps able to retain more information about each residue) and correspondingly slower. It is likely that these models encode some high level information about sequences that remains uncaptured by NEAR, but our results indicate that this extra information does not improve prefilter performance. Furthermore, NEAR improves speed ∼10x over HMMER's MSV filter when run on a system with a commodity GPU. These results collectively demonstrate the efficacy of NEAR in accurately ranking hits.

Though NEAR shows promise as a fast and accurate filter for pHMM search, we have not created an integrated tool connecting NEAR with HMMER or any other pHMM software. Instead, we have focused on designing an effective model architecture and training strategy, and also performing experiments to explore NEAR's viability as a pre-filter. Counter-intuitively, NEAR works well as a prefilter not by compressing data, but by expanding data. NEAR transforms simple sequence strings into large and complex sequences of high dimensional vectors, enabling us to harness the wealth of research and software available in the field of vector search. An important concern regarding NEAR's utility as an alignment filter is that by representing residues as high dimensional vectors, NEAR has limited utility on large datasets; precisely where fast filters are most important. Using a quantized search index allows for a modest reduction in memory and compute costs, but a much greater reduction is necessary for NEAR to be a viable prefilter method for large sequence databases. Advances to methods for sparse sequence representation, perhaps based on prediction of which residues are important for search, will be vital for improved scalability.

Acknowledgments

We thank Tim Anderson, Jack Roddy, and Genevieve Krause for helpful discussions during development of software and benchmarks. We also gratefully acknowledge the computational resources and expert administration provided by the University of Montana's Griz Shared Computing Cluster (GSCC), CyVerse's External Collaborative Partnership program (provided by Tyson Swetnam), and the high performance computing (HPC) resources supported by the University of Arizona TRIF, UITS, and Research, Innovation, and Impact (RII) and maintained by the UArizona Research Technologies department.

References

1. Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.
2. Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, and Michal Linial. ProteinBERT: a universal deep-learning model of protein sequence and function. *Bioinformatics*, 38(8):2102–2110, 2022.
3. C Titus Brown and Luiz Irber. sourmash: a library for MinHash sketching of DNA. *Journal of open source software*, 1(5):27, 2016.
4. Benjamin Buchfink, Klaus Reuter, and Hajk-Georg Drost. Sensitive protein alignments at tree-of-life scale using DIAMOND. *Nature methods*, 18(4):366–368, 2021.
5. Christiam Camacho, George Coulouris, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer, and Thomas L Madden. BLAST+: architecture and applications. *BMC bioinformatics*, 10:1–9, 2009.
6. Quentin Carradec, Eric Pelletier, Corinne Da Silva, Adriana Alberti, Yoann Seeleuthner, Romain Blanc-Mathieu, Gipsi Lima-Mendez, Fabio Rocha, Leila Tirichine, Karine Labadie, et al. A global ocean atlas of eukaryotic genes. *Nature communications*, 9(1):373, 2018.
7. Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong

Wang. SPANN: Highly-efficient billion-scale approximate nearest neighbor search. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021.
8. Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.
9. Sean R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, 1998.
10. Sean R Eddy. Accelerated profile HMM searches. *PLoS Comput Biol*, 7(10):e1002195, 2011.
11. RC Edgar. USEARCH and UCLUST algorithms. *Bioinformatics*, 26:2460–2461, 2010.
12. Robert C Edgar, Brie Taylor, Victor Lin, Tomer Altman, Pierre Barbera, Dmitry Meleshko, Dan Lohr, Gherman Novakovsky, Benjamin Buchfink, Basem Al-Shayeb, et al. Petabase-scale sequence alignment catalyses viral discovery. *Nature*, 602(7895):142–147, 2022.
13. Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rihawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. ProtTrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing. *arXiv preprint*, 2021.
14. William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019.
15. Martin C Frith. A new repeat-masking method enables specific detection of homologous sequences. *Nucleic acids research*, 39(4):e23–e23, 2011.
16. Martin C Frith, Michiaki Hamada, and Paul Horton. Parameters for accurate genome alignment. *BMC bioinformatics*, 11:1–14, 2010.
17. Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint*, 2017.
18. George Glidden-Handgis and Travis J Wheeler. WAS IT A MATch I SAW? Approximate palindromes lead to overstated false match rates in benchmarks using reversed sequences. *bioRxiv*, pages 2023–06, 2023.
19. Michael Gribskov, Andrew D McLachlan, and David Eisenberg. Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences*, 84(13):4355–4358, 1987.
20. Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR, 2020.
21. 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.
22. Masajiro Iwasaki and Daisuke Miyazaki. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *arXiv preprint*, 2018.
23. Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.
24. Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.
25. Kevin Karplus, Christian Barrett, and Richard Hughey. Hidden Markov models for detecting remote protein homologies. *Bioinformatics (Oxford, England)*, 14(10):846–856, 1998.
26. Szymon M Kiełbasa, Raymond Wan, Kengo Sato, Paul Horton, and Martin C Frith. Adaptive seeds tame genomic sequence comparison. *Genome research*, 21(3):487–493, 2011.
27. Dhananjay Kimothi, Akshay Soni, Pravesh Biyani, and James M Hogan. Distributed representations for biological sequence analysis. *arXiv preprint*, 2016.
28. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
29. Genevieve R. Krause, Walt Shands, and Travis J. Wheeler. Sensitive and error-tolerant annotation of protein-coding DNA with BATH. *bioRxiv*, 2024.
30. Anders Krogh, Michael Brown, I Saira Mian, Kimmen Sjölander, and David Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of molecular biology*, 235(5):1501–1531, 1994.
31. Sewon Lee, Gyuri Kim, Eli Levy Karin, Milot Mirdita, Sukhwan Park, Rayan Chikhi, Artem Babaian, Andriy Kryshtafovych, and Martin Steinegger. Petascale homology search for structure prediction. *bioRxiv*, 2023.
32. Eli Levy Karin, Milot Mirdita, and Johannes Söding. MetaEuk—sensitive, high-throughput gene discovery, and annotation for large-scale eukaryotic metagenomics. *Microbiome*, 8:1–15, 2020.
33. Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.
34. Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.
35. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint*, 2013.
36. Sejal Modha, David L Robertson, Joseph Hughes, and Richard J Orton. Quantifying and cataloguing unknown sequences within human microbiomes. *Msystems*, 7(2):e01468–21, 2022.
37. Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
38. William R. Pearson. An Introduction to Sequence Similarity ("Homology") Searching. *Current Protocols in Bioinformatics*, 6 2013.
39. Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
40. Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

41. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.
42. Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15):e2016239118, 2021.
43. Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome research*, 31(11):2080–2094, 2021.
44. Konstantin Schütze, Michael Heinzinger, Martin Steinegger, and Burkhard Rost. Nearest neighbor search on embeddings rapidly identifies distant protein relations. *Frontiers in Bioinformatics*, 2:1033775, 2022.
45. Kihyuk Sohn. Improved deep metric learning with multi-class N-pair loss objective. *Advances in neural information processing systems*, 29, 2016.
46. Martin Steinegger and Johannes Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028, 2017.
47. Baris E Suzek, Yuqi Wang, Hongzhan Huang, Peter B McGarvey, Cathy H Wu, and UniProt Consortium. UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, 31(6):926–932, 2015.
48. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
49. Yiren Wang and Fei Tian. Recurrent residual learning for sequence classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 938–943, 2016.