Deep embedding and alignment of protein sequences

Felipe Llinares-López, Quentin Berthet, Mathieu Blondel,
Olivier Teboul and Jean-Philippe Vert*

Google Research, Brain team, Paris, France

November 15, 2021

Abstract

Protein sequence alignment is a key component of most bioinformatics pipelines to study the structures and functions of proteins. Aligning highly divergent sequences remains, however, a difficult task that current algorithms often fail to perform accurately, leaving many proteins or open reading frames poorly annotated. Here, we leverage recent advances in deep learning for language modelling and differentiable programming to propose DEDAL, a flexible model to align protein sequences and detect homologs. DEDAL is a machine learning-based model that learns to align sequences by observing large datasets of raw protein sequences and of correct alignments. Once trained, we show that DEDAL improves by up to two- or three-fold the alignment correctness over existing methods on remote homologs, and better discriminates remote homologs from evolutionarily unrelated sequences, paving the way to improvements on many downstream tasks relying on sequence alignment in structural and functional genomics.

Introduction

Sequence alignment is a key component of bioinformatics pipelines to study the structures and functions of proteins, and to annotate open reading frames (ORF) in newly sequenced genomes and metagenomes [1]. Indeed, aligning two protein sequences allows identifying homologous sequences with known structure or function, and regions of similarity that may be the result of evolutionary, structural or functional relationships. Jointly aligning multiple sequences further gives information about evolutionary constraints and patterns of co-evolution along the sequence, which is for example useful for 3D structure inference [2–4]. It is remarkable that algorithms for pairwise sequence alignment have basically remained unchanged since their invention in the 1980's and 1990's, in particular the Smith-Waterman (SW) algorithm to find the best local alignment between two sequences in quadratic time with dynamic programming [5] and faster heuristics like BLAST [6] or FASTA [7] which are also the first steps of more advanced methods based on multiple sequence alignment such as PSI-BLAST [8]. However, in spite of their immense popularity and importance for downstream applications, these algorithms often produce erroneous alignments or fail to detect homology, particularly for sequences with low similarity [9]. This leaves a sizeable fraction of predicted ORF in genomics and metagenomics projects without annotation [10], while alignment errors can in turn result in wrong structural or functional annotations. Improving pairwise sequence alignment algorithms, particularly for divergent sequences, could directly benefit a number of downstream tasks.

The SW algorithm formulates the problem of finding the correct alignment between two sequences as a search for the best-scoring path in a graph. Each candidate path represents a possible alignment and the score of a path depends on user-defined parameters. These are the

*Correspondence: jpvert@google.com

costs of starting or extending a gap in the alignment, and the substitution score of aligning each position of the first sequence to each position of the second one, which usually depends on the amino acids at both positions. This formulation, which leads to efficient algorithms to find or approximate the best-scoring alignment, has at least two weaknesses that may lead to possible errors in the alignment found. First, it lacks flexibility in the formulation of the score of a path, by enforcing for example that the cost of opening or extending a gap is the same wherever in the sequences, or by enforcing that the substitution score of aligning two positions only depends on the amino acids at those positions. Allowing more flexible parameterizations may give more opportunities to the algorithm to optimize a relevant score; for example, [11] show the benefits of making the substitution score between two positions in protein sequences depend not only on the amino acids at those positions, but also on predicted structural properties, at the cost of increasing the number of parameters of the model. Second, the best alignment returned by the algorithm strongly depends on the choice of the parameters [12–18], and even for simple models where the substitution score only depends on the amino acids to be aligned, it is well known that there is no universally "good" substitution scores that lead to good alignments for all protein pairs [19, 20]. This raises the question of how to optimally choose adequate scoring parameters for a given pair of proteins.

In this work, we propose DEDAL (*Deep Embedding and Differentiable ALignment*), an algorithm for pairwise sequence alignments that addresses both issues. DEDAL builds on top of the standard SW algorithm to efficiently find an optimal alignment between two sequences, but provides a flexible parameterization of the scoring function used by the SW algorithm that adapts to each sequence pair and each position in each sequence. The parameterization is automatically learned during a training phase from a set of sequence pairs with known alignments, and a large set of raw protein sequences. It relies both on recent advances in deep learning language models which embed discrete sequences in a continuous space and are automatically trained on a large corpus of raw sequences, and also a parameterization of the SW algorithm (gap and substitution parameters) as a function of the continuous embedding. To train DEDAL, we propose a smoothed variant of the SW algorithm, to make the alignment solution a continuous and differentiable function of the scoring parameters. Given a set of sequence pairs with known correct alignment, we then tune automatically the various parameters of the model by end-to-end gradient-based optimization to minimize the alignment errors. Once trained, DEDAL produces gap and substitution scoring matrices computed specifically for each new pair of sequences. In addition, the gap and substitution scores are contextual: for each pair of positions, they depend on the full sequences to be aligned. The optimal alignment is then computed with a standard SW algorithm using those parameters. We show that DEDAL can be trained efficiently on modern hardware with accelerators. Once trained, we demonstrate that DEDAL improves by up to two- or three-fold the quality of the alignment predicted for remote homologs compared to standard SW, and produces an alignment score that more accurately detects remote homology.

Related work. The fact that the solution of the SW algorithm depends on the scoring parameters has been studied in detail in the context of the so-called parametric sequence alignment problem, which aims to describe the set of possible solutions as a function of the parameters used [12–18]. This approach, however, is only feasible for simple models with up to 2 or 3 parameters that vary. Conversely, the idea to search for parameters that reproduce a set of given, correct alignments, was tackled by [11, 21–23] using various optimization techniques, but these works focus on simple parameterizations where a fixed-size substitution matrix and one or two gap parameters are optimized. It is proposed in [24] to exploit known alignments to help train deep embedding models with a differentiable loss, however this model does not produce a sequence alignment once trained. The closest work to ours is the DeepBlast model of [25], who also propose to learn a deep language model for proteins with a differentiable alignment module, however the model has a simpler model for scores (linear instead of affine), and only

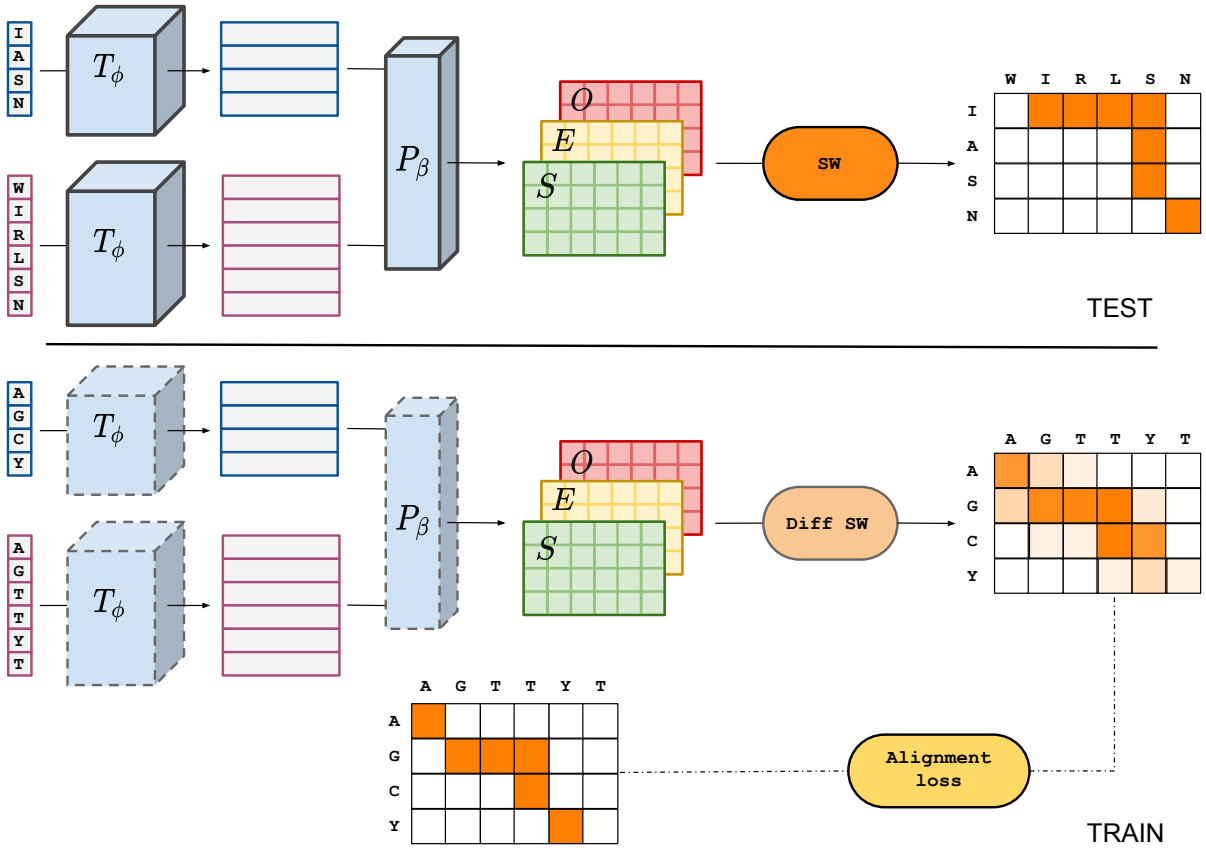


Figure 1: Overview of DEDAL. At test time (top), DEDAL aligns two sequences x and y by running the SW local alignment algorithm with position-specific parameters (gap open O, gap extend E, substitution scores S) that depend on the input sequences through a transformer encoding T_ϕ of each sequence followed by a parameterizer P_β that transforms the continuous representations into matrices of parameters needed by SW. Both T_ϕ and P_β depend on parameters ϕ and β which are learned at train time from a large corpus of raw sequences to learn the language model T_ϕ, combined with pairs of sequences with known alignment to learn jointly T_ϕ and P_β by end-to-end optimization with a differentiable variant of SW and a specific loss function (bottom).

produces global alignments. More recently, concurrently and independently from this work, [26] presented a differentiable version of SW for multiple sequence alignment, albeit with a simple convolution layer instead of a full language model to embed the sequences, and simpler position-independent gap penalty. Moreover, they focus on contact and structure prediction tasks, as opposed to directly investigating alignment and homology detection performance as done here.

Results

Pairwise local alignment or protein sequences with DEDAL

We introduce DEDAL, a trainable algorithm for accurate pairwise local alignment of protein sequences (Figure 1). DEDAL aligns sequences by computing substitution scores and gap penalties that are specific to the sequences being aligned (Figure 1, top). To this end, DEDAL depends on parameters which are automatically tuned before inference during a training phase (Figure 1, bottom).

For sake of clarity, let us first describe how DEDAL works once it is trained and ready to be used to align sequences. In order to align two sequences x and y and score the resulting

alignment, DEDAL simply uses the standard SW algorithm for pairwise local alignments, but with gap open, gap extend and substitution score matrices which are computed specifically from x and y. To do so, a deep learning-based transformer encoder network T_ϕ with parameters ϕ [27] is first used to independently obtain a continuous representation of each of these sequences. In this representation, each residue of each sequence is mapped to a vector in a vector space of a fixed dimension (we use $d = 768$ in our experiments). Crucially, these embeddings are *contextual*, that is, the embedding of each residue encodes information not only about the amino acid present at that position but also about all other residues in the sequence, as well as their relative arrangement. This allows DEDAL to be highly flexible in the way sequences are represented, opting for a data-driven approach towards incorporating contextual information over hard-coded rules. Next, DEDAL computes a substitution score as well as gap open and gap extend penalties for each pair of residues from the sequences to be aligned, computed from their respective vector representations by a parameterizer function P_β, which depends on parameters β. Finally, the standard SW algorithm is used to compute an optimal alignment and score it, using the substitution scores, gap open and gap extend penalties computed at the previous step. In other words, DEDAL relies on the SW algorithm to align sequences and score the alignment, but provides a very flexible framework to parameterize the SW algorithm; in particular, the substitution score, gap open and gap extend penalties are specific to each pair of positions in the two input sequences, and depend on the full sequences through the contextual embedding of the transformer encoder and the parameterizer.

DEDAL thus depends on parameters ϕ for the transformer encoder T_ϕ, and β for the parameterizer P_β. These parameters are tuned automatically during a training phase, where we provide DEDAL with a large set of ~ 30 million non-redundant protein sequences from UniRef50 [28] to train the transformer encoder, as well as a set of pairs of homologous sequences with curated correct alignment extracted from the set of ~ 1.2 million sequences organized in ~ 19 thousand families in the Pfam-A seed database [29] to jointly train the transformer encoder and the parameterizer. More precisely, using these datasets, DEDAL tunes its parameters by end-to-end gradient-based optimization to minimize a loss function combining three tasks: 1) a so-called *masked language modelling* task on the UniRef50 dataset, which is a classical way to tune the parameters of a transformer encoder [30, 31]; 2) an *homology detection* task, where we train DEDAL so that it can discriminate homologous from non-homologous pairs in Pfam-A seed from the alignment score; 3) a *learning to align* task, where we train DEDAL to produce the correct alignment on homologous pairs from Pfam-A seed with known correct alignment. In order to solve the second and third tasks by gradient-based optimization, we implement a novel differentiable variant of the SW algorithms that allows to backpropagate not only the alignment score but also the error between the predicted and true alignments to the parameters ϕ and β. Since Pfam sequences represent protein domains, which therefore tend to be aligned from start to end within a given family, we extend the sequences when possible upstream and downstream of the domain so that DEDAL is trained to produce a correct *local* alignment. We refer the reader to the Methods section for more details about the model and the data.

DEDAL accurately aligns homologous sequences

We first assess the ability of DEDAL to accurately align homologous sequences. Since DEDAL is trained on a set of known correct alignments, we must evaluate its performance on sequences not seen at train time. We therefore split the Pfam sequences into two non-overlapping sets. The first one is used to train the model and to choose hyperparameters, and the second one to assess its performance. We consider two ways to make the split: 1) split sequences uniformly at random in Pfam, so that sequences in the test set come from families seen at train time (we call this setting the *in-distribution* setting), and 2) split clans between training and test sets, so that sequences in the test set come from clans that are not seen at train time (we call this setting the *out-of-distribution* setting). Obviously the out-of-distribution setting is more challenging, since

it requires the model to generalize across clans. It mimics the situation where we want to align sequences from unknown domains. The in-distribution setting, on the other hand, mimics the common situation where we want to align sequences from known Pfam domains. We measure the quality of a predicted alignment by the F_1 score of the prediction on the test set, and stratify the performance by percent identity (PID) of the sequences in the correct alignment, since it is well known that the difficulty of aligning sequences increases when PID decreases. As a baseline, we compare DEDAL to the SW algorithm ran with a fixed PFASUM60 substitution matrix and an affine gap penalty model with gap open=16 and gap extend=2. We chose this configuration as the best among more than $1,000$ combinations of substitution matrices in the BLOSUM [32], VTML [33, 34] and PFASUM [20] families, and gap open and extend parameters (see Methods).

Figures 2a and 2b summarize the alignment performance of DEDAL and of the baseline, respectively in the in-distribution and out-of-distribution settings. A breakdown of the F_1 score into precision and recall is shown in Supp. Figures S7 and S8, respectively. We see that DEDAL significantly outperforms the baseline both in-distribution and out-of-distribution. The difference is particularly strong in-distribution (with an average relative improvement of 25% over baselines, from $F_1 = 0.673$ to $F_1 = 0.843$), which confirms the benefit of training DEDAL on Pfam families in order to then align new protein sequences within these families. Strikingly, the gap between DEDAL and baselines widens as the PID decreases, with a relative improvement of up to 219% in the hardest setting (PID ≤ 0.1), showing that DEDAL is able to provide good quality alignments ($F_1 = 0.520$) even between very remote homologs, while the baseline can not ($F_1 = 0.163$). Interestingly, a similar pattern is visible on the out-of-distribution setting, where DEDAL outperforms the baselines by 5% on average (from $F_1 = 0.709$ to $F_1 = 0.742$), and by 89% on the remote homologs with PID < 0.1 (from $F_1 = 0.165$ to $F_1 = 0.313$). This suggests that DEDAL learns a model of protein sequence similarity that extends beyond the biological subspace it sees at train time, and that DEDAL is particularly useful is difficult situations of remote homology, where standard methods perform poorly.

We then compare DEDAL and the baseline on their ability to align Pfam domains, as opposed to the extended domains that we use to train DEDAL. While the domain sequences are the same in both cases, the alignments are very different since two domain sequences in a Pfam family can generally be aligned from the beginning to the end, thus making the correct alignment closer to a global alignment than a local one. We show the results in this setting in Figures 2c and 2d. Compared to the previous setting, we see that the performances of all methods tend to be better, which can be explained by the fact that the number of wrong candidate alignments is reduced when we focus on the domain sequence that must be aligned. Besides the overall improvement of all methods, DEDAL still significantly outperforms the baseline, particularly in-distribution and particularly in the low PID regime (with, e.g., a relative F_1 increase of 173% for in-distribution domains with PID ≤ 0.1, from $F_1 = 0.201$ to $F_1 = 0.551$). We also notice that for "easy" in-distribution cases (PID ≥ 0.5), the baseline and DEDAL reach similarly high accuracy, suggesting that the baseline suffers particularly when it needs to find local alignments as opposed to global ones, even in the high PID regime, while DEDAL behaves equally well in both settings.

In order to clarify why DEDAL behaves so differently from a standard SW algorithm with a fixed substitution matrix and gap parameters, we now focus on a particular pairwise alignment between two domain sequences of Pfam-A seed (accession PF00048): a C-C motif chemokine from western lowland gorilla (UniProt A0A2I2ZUR5) and a C-X-C motif chemokine ligand 14 from northern mallard (UniProt U3IBZ2). Figure 3 summarizes the alignments produced by DEDAL and the baseline (using a PFASUM substitution matrix), and the parameters of both models. Note that this alignment is not part of DEDAL's training set. As seen in Figure 3.a, the sequences have only 8 conserved residues in the Pfam-A seed curated alignment, meaning that this is a difficult case (PID=0.12). Indeed, using SW with a PFASUM substitution matrix gives

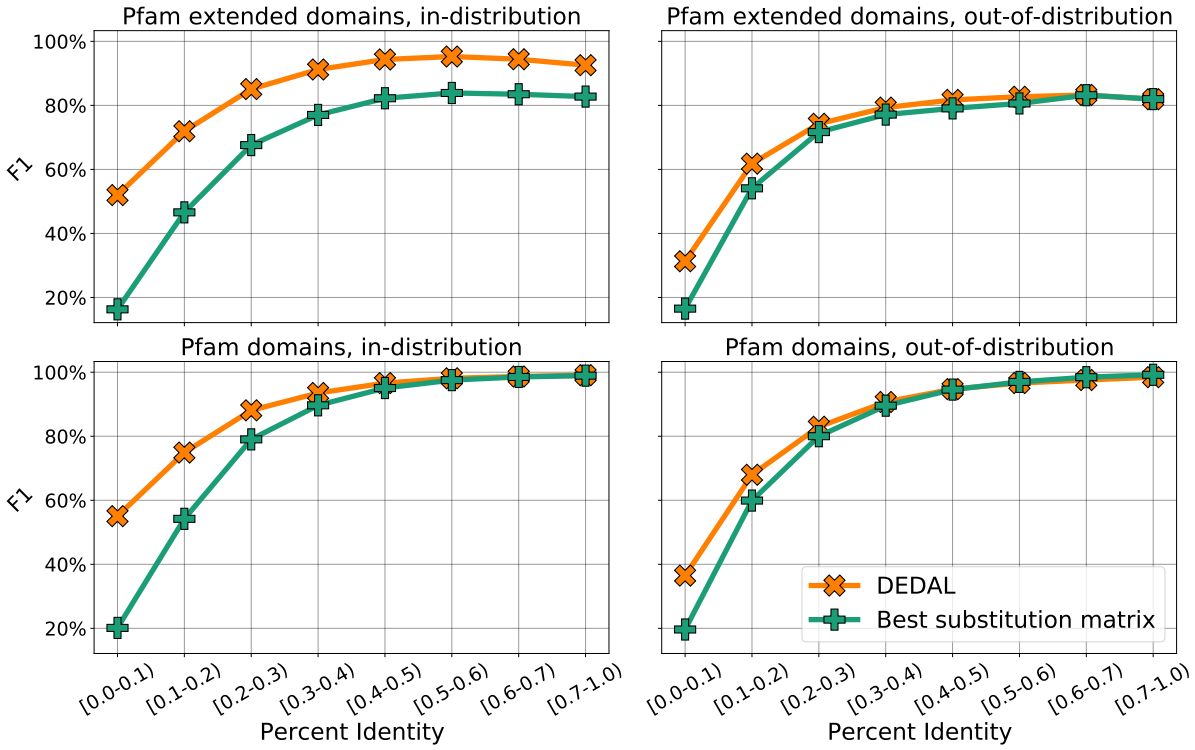


Figure 2: Alignment F_1 score of DEDAL and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows).

an alignment where no residue is correctly aligned ($F_1 = 0$), however we see that using DEDAL gives an almost perfect alignment where only 7 residues are wrongly aligned, and in all cases with an error of only one residue in distance ($F_1 = 0.88$). To better understand the difference between PFASUM and DEDAL, we plot in Figure 3.b and 3.c, respectively, the SW parameters for PFASUM (substitution scores) and for DEDAL (substitution scores, gap open and gap extend penalties). We see that the substitution score matrix produced by DEDAL is much smoother spatially than the one of PFASUM, highlighting the benefit of context dependent embeddings of residues. We also see that DEDAL clearly picks the importance of aligning cysteine (C) residues, which are very conserved since they form disulfide bonds which stabilize the 3D structure of the domain (there are four cysteines in each sequences, respectively at positions 3, 4, 27, 43 in gorilla and 3, 5, 29, 49 in mallard, corresponding to the 16 positions in DEDAL's substitution matrix with highest score). PFASUM, on the other hand, has high scores not only to match the cysteines, but also to match the two tryptophans (W) at respective positions 21 and 49 in the gorilla sequence, and 38 and 63 in the mallard sequence. While tryptophans are usually very conserved, and have therefore a large substitution score in standard substitution matrices such as PFASUM, it is interesting that in this particular case they should not be aligned according to the Pfam-A seed curated alignment, and DEDAL correctly predicts it by producing a small substitution score. Finally, we note that DEDAL's gap open and extend penalties are far from uniform in the matrix, with in particular strong penalties to open and extend gaps near the cysteines, suggesting that DEDAL has learned that mutations are more likely than insertions or deletions near a cysteine involved in a disulfide bond.

DEDAL accurately detects remote homologs

Next, we seek to determine if DEDAL's ability to align homologous sequences accurately also implies that the alignment score it computes is effective to detect homology. To this end, we

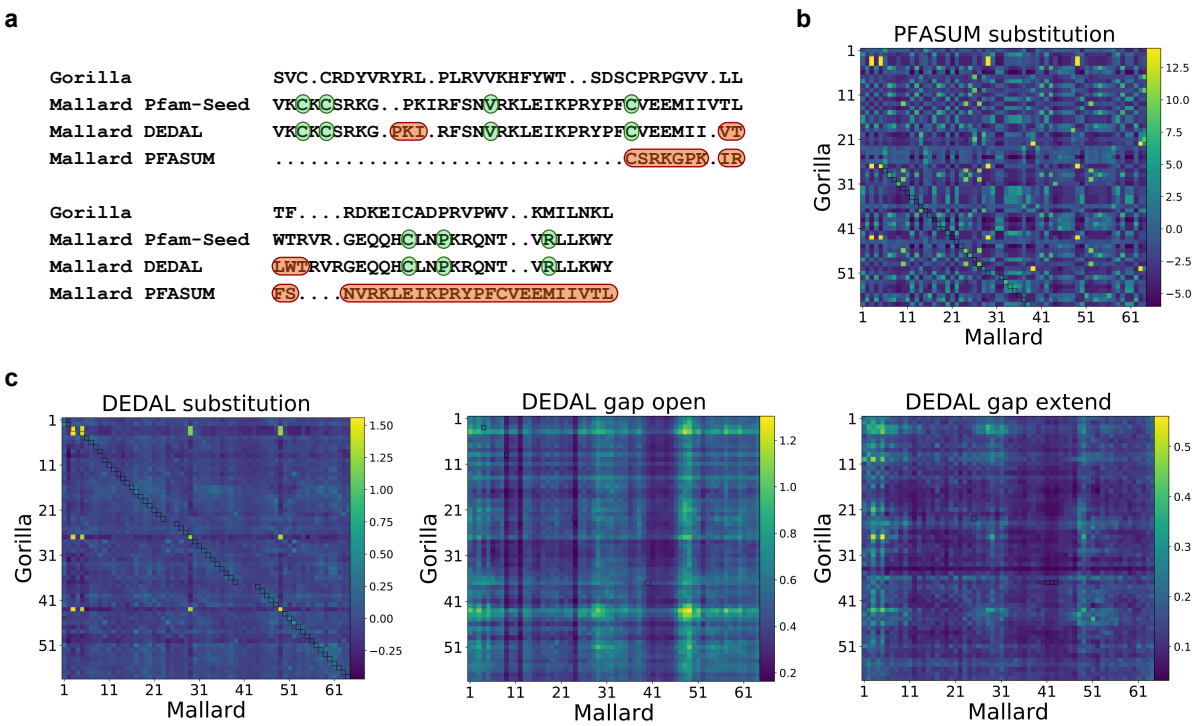


Figure 3: Example of pairwise alignment of two protein domain sequences from Pfam-A seed, a C-C motif chemokine from western lowland gorilla (UniProt A0A2I2ZUR5) and a C-X-C motif chemokine ligand 14 from northern mallard (UniProt U3IBZ2). **a.** Alignment respectively from the curated Pfam-A seed database (second row), predicted by DEDAL (third row), and predicted with a PFASUM60 substitution matrix (fourth row). We show all residues in both sequences for the Pfam-A seed and DEDAL alignments, but not the unaligned residues upstream and downstream of the alignment in the mallard sequence for PFASUM. Residues highlighted in green correspond to correctly aligned conserved residues, while those in red correspond to discrepancies between a predicted alignment and the Pfam-A seed one. **b.** Substitution scores between all pairs of residues from the PFASUM substitution matrix. **c.** SW parameters predicted by DEDAL.

follow the same experimental setup as when probing alignment performance, namely considering pairs of Pfam extended domains or pairs of Pfam domains, on the one hand, and considering pairs of candidate homologous sequences from families seen at train time (in-distribution setting) or from clans not seen at train time (out-of-distribution setting), on the other hand. For each of these settings, we measure the area under the precision-recall curve (AUPRC) when predicting whether a pair of sequences belongs to the same Pfam family or not based on the SW score, correcting the alignment score for sequence length as usually done when computing alignment E-values to assess homology (see Methods). We again stratify AUPRC values according to the PID of related sequence pairs (positive class), with unrelated pairs (negative class), whose ground-truth PID is unknown, being included in all bins.

The results of this experiment are summarized in Figure 4. We also show in Supp. Figure S9 an alternative manner to evaluate performance, using the area under the receiver operating characteristic curve (AUROC). We see that, unsurprisingly, all methods are largely successful in detecting homology for sequence pairs with moderate-to-large PID (≥ 0.2) in all settings. Interestingly, DEDAL is markedly better at detecting homology of highly divergent sequences across all four settings. For the smallest PID bin (< 0.1), DEDAL improves in-distribution performance by 154% (from AUPCR=0.389 to AUPCR=0.988) and 97% (from AUPCR=0.501 to AUPCR=0.987) for Pfam domains and extended domains, respectively. Once again, the

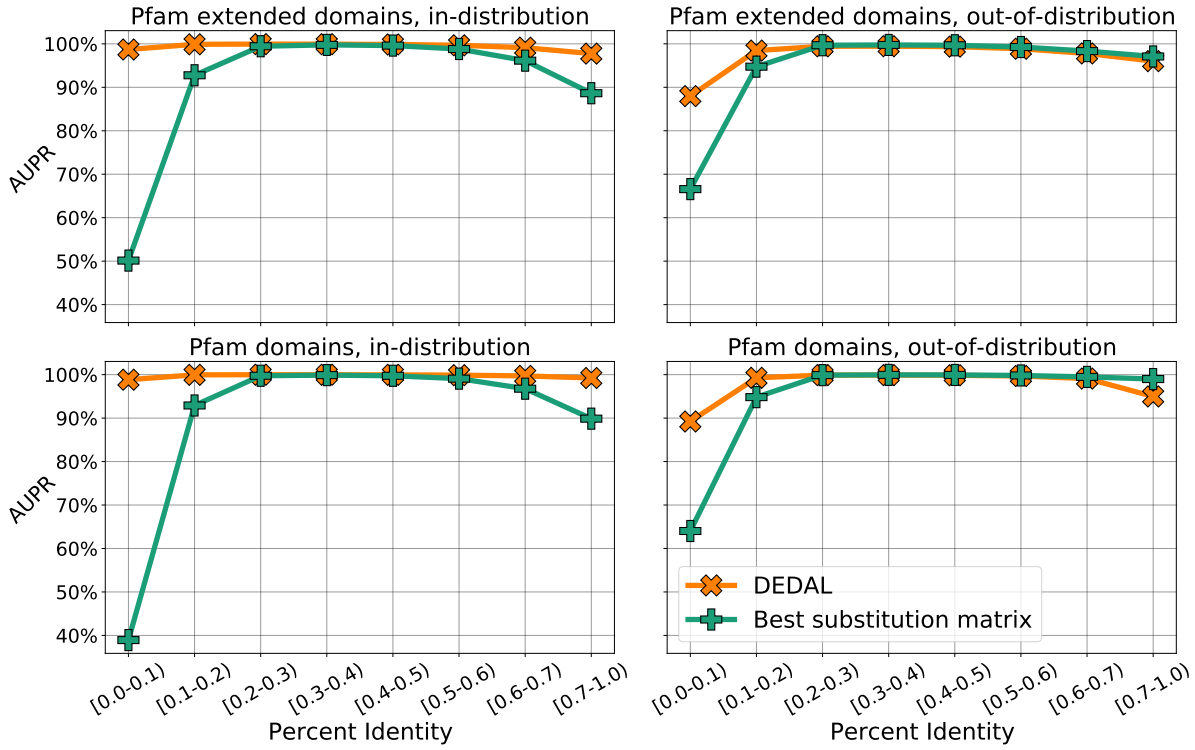


Figure 4: Homology detection AUPRC of DEDAL and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows).

performance of DEDAL degrades in the out-of-distribution split while remaining significantly superior to the baselines, boosting AUPRC by 39% (from AUPCR=0.640 to AUPCR=0.892) and 32% (from AUPCR=0.666 to AUPCR=0.880) for domains and extended domains, respectively. Interestingly, DEDAL only seems to be outperformed by a small yet non-negligible margin in the largest PID bin (≥ 0.7) of the out-of-distribution split, where its AUPRC is 4% (AUPCR=0.990 vs. AUPCR=0.950) and 1.2% (AUPCR=0.971 vs. AUPCR=0.960) inferior relative to that of the substitution matrices. A possible explanation for this phenomenon is that, as shown in Supp. Figure S6, sequence pairs in that PID range are exceedingly scarce in DEDAL's training dataset, representing less than 1% of all sequence pairs. Upsampling schemes might be investigated as a potential way to alleviate this shortcoming.

Discussion

Using recent advances in deep language models with transformers and novel differentiable alignment modules, we showed that DEDAL learns a continuous representation of protein sequences that, combined with the SW algorithm, leads to more accurate pairwise sequence alignment and homology detection than using the SW algorithm with fixed substitution matrices and gap penalties. The improvement is particularly striking in hard cases, where we want to align remote homologs with limited sequence identity, and we showed that the model learned by DEDAL on some sequence space generalizes well to new clans, suggesting that DEDAL learns universal biological properties not easily captured by standard substitution matrices and affine gap penalties.

Since DEDAL incorporates a number of design choices, such as the architecture of the model or the strategies to train it, one may wonder which aspects are the most critical to explain its success. To address that question, we performed an ablation study where we systematically

evaluated the effect of various design choices on the performance of DEDAL (Supplementary Results S2.1). In short, we found that replacing our rich parameterization of position-specific gap open and extend parameters by a simpler model where the gap open and extend parameters are position independent does not affect the performance of DEDAL much, achieving slightly superior alignment F_1 scores in the in-distribution split but marginally inferior results for out-of-distribution sequence pairs. Another simplification would be to replace our position-specific affine gap penalty by a position-specific linear gap penalty, as for example used in DeepBLAST [25], which, however, we found leads to a small performance drop for remote homologs that is most pronounced in the out-of-distribution split.

On the technical side, we explored two approaches to create a differentiable SW alignment module, needed to train the parameters of DEDAL in the "learning to align" task, using either a smoothing technique [35] or a perturbation technique [36]; we found no significant difference between both in terms of performance, and implemented the one based on perturbations in the final DEDAL model. Regarding the set of alignments used to train DEDAL, we found that it is beneficial to use Pfam extended domains instead of Pfam domains when we want DEDAL to be able to predict accurate local alignments. Regarding the strategy to train end-to-end jointly the transformer and parameterizer, we found that this is indeed significantly better than a more classical, two-step strategy that would first train a transformer encoder on the masked language modelling task, and second the parameterizer on the "learning to align" task by keeping the transformer fixed. This suggests that a universal language model, such as the one presented by [31], is not sufficient and should be at least fine-tuned for optimal performance in alignment. Finally, we assessed the benefit of context-dependent embeddings by simply training a model where the substitution cost is constrained to depend only on the amino acids to be aligned; unsurprisingly, we observed a strong drop in performance for this model, reaching about the same performance as the best-performing substitution matrix in the literature. All in all, our ablation study indicates that DEDAL is robust to changes in several technical aspects of the model, but that the main idea to jointly train a deep language model and a flexible parameterizer for SW is key to its performance.

This work opens several research directions for the future. First, as pure language models are increasingly used for a variety of downstream tasks, it is interesting to explore if adding alignment information to supervise language models, as DEDAL does, may also benefit other downstream tasks beyond sequence alignment. In preliminary experiments, we did not observe significant differences between the transformer encoder trained by DEDAL and one purely trained on a language modelling task on the TAPE benchmark [37], a collection of problems to assess the relevance of protein embedding models for various downstream tasks (see Supplementary Results S2.2), but believe there is room for improvement. Second, it will be interesting to extend DEDAL to the multiple alignment setting (MSA), either by combining pairwise alignment operations, as proposed by [26], which could directly benefit from DEDAL, or by directly embedding multiple non-aligned sequences and creating a differentiable MSA module. Third, given the accuracy of DEDAL's alignment and ability to capture homology, it will be interesting to develop fast approximations based, e.g., on fast similarity search in continuous spaces, in order to allow accurate homology search and alignment in large databases.

Methods

Data

To train the transformer encoder with a masked language modelling task, we used $30,162,111$ cluster representative sequences from the March 2018 release of the UniRef50 database [28]. These representatives provide a non-redundant yet complete coverage of UniProtKB [38], obtained by clustering sequences with at least 50% percent identity and 80% overlap with the

cluster's longest sequence using MMseqs2 [39]. A representative for each cluster was chosen accounting for factors such as quality of the UniProtKB entry and availability of annotations, among others. Given the relatively low redundancy between UniRef50 representative sequences, sequences were allocated to train or test uniformly at random as in [31], resulting in training and test sets of size $27,052,653$ and $3,019,006$, respectively. A small validation set of $90,452$ sequences was further reserved for hyperparameter tuning. For computational efficiency, we limit the length of sequences fed to the model for this task to at most $1,023$ amino acids. Longer sequences are cropped to length $1,023$ with a uniformly distributed offset sampled on-the-fly at training and evaluation time.

To create pairs of homologous sequences with known alignment, we collected $19,179$ manually curated multiple sequence alignments (MSAs) from release 34.0 of the Pfam-A seed database [29], grouped into $12,452$ evolutionarily-related clans, and spanning a total of $1,223,021$ sequences. We kept only the $1,198,870$ sequences (98% of the total) of length smaller than 512. We define a pair of homologous sequences as a pair of sequences in the same MSA, from which we extract a correct pairwise alignment. Any two sequences not in the same MSA are deemed as not homologous in the homology detection task. From this set of Pfam domain sequences, we also build a set of Pfam extended domain sequences, by adding flanking sequences with random lengths upstream and downstream of each domain sequence, following the existing flanking regions of the domain in the UniProtKB protein sequence database (see Supplementary Methods S1.1). This ensures that the correct alignment between two homologous extended domains usually does not span the full sequences. For both Pfam domains and extended domains, we split the full dataset into five disjoint splits: train (80%), in-distribution validation (2.5%), out-of-distribution validation (2.5%), in-distribution test (7.5%) and out-of-distribution test (7.5%). To this end, we first select 255 and 955 out of the $12,452$ Pfam clans uniformly at random to form the out-of-distribution validation and out-of-distribution test, respectively. The number of clans was chosen to reach the target sample sizes of 2.5% and 7.5% for each split. Finally, we divide each of the remaining $10,493$ clans uniformly at random to form the train, in-distribution validation and in-distribution test splits.

DEDAL model

DEDAL is a parametric model that transforms a pair of sequences onto three matrices of gap open, gap extend penalties and substitution costs, respectively, and then uses the SW algorithm to align the two initial sequences using these matrices of parameters. Each protein sequence is seen as a sequence of tokens in a 27-letter alphabet (22 amino acids including pyrrolysine (O) and selenocysteine (U), 3 ambiguous codes B, Z and X, and 2 special tokens `<EOS>`, which we append to each sequence, and `<MASK>`, which is used for masked language modelling only). Each such sequence is mapped to a sequence of vectors by a transformer encoder neural network similar to the smallest transformer-based model of [31] which uses $L = 6$ transformer encoder layers with $h = 12$ heads per layer and embeddings of dimension $d = 768$. Each transformer encoder layer combines multi-head self-attention, position-wise multilayer perceptron, dropout and layer normalization. From the output of the transformer, we compute the substitution score between two positions as a bilinear function of their vector representations, and the gap and open penalties as the softplus function applied to a bilinear function of their vector representations to ensure they are nonnegative. An in-depth description of the DEDAL model as well as of its architecture and hyperparameters can be found in Supplementary Methods S1.2.

Differentiable SW alignment

Given two sequences $x = x_1 \dots x_n$ and $y = y_1 \dots y_m$ of amino acids, an alignment π is a subset of distinct positions in x and y that are matched in increasing order. Let us denote by $\Pi(x, y)$ the set of possible alignments between x and y, including the empty one. Given three $n \times m$ matrices

S (substitution scores), O (gap open) and E (gap extend penalties), the local alignment score $s(\pi)$ of a candidate alignment π is defined as the sum of the substitution scores over matched positions, and gap penalties defined as the gap open penalty where a gap starts and gap extend penalties where it continues. The SW algorithm finds the maximum scoring alignment with the dynamic programming recursion over the $(n+1) \times (m+1)$ matrices M, X and Y defined for any $i = 1, \dots, n$ and $j = 1, \dots, m$ by $M_{i,0} = M_{0,j} = X_{i,0} = X_{0,j} = Y_{i,0} = Y_{0,j} = -\infty$ and:

$$\begin{cases} M_{i,j} &= S_{i,j} + \max(0, M_{i-1,j-1}, X_{i-1,j-1}, Y_{i-1,j-1}) \,, \\ X_{i,j} &= \max(M_{i,j-1} - O_{i,j}, X_{i,j-1} - E_{i,j}) \,, \\ Y_{i,j} &= \max(M_{i-1,j} - O_{i,j}, X_{i-1,j} - O_{i,j}, Y_{i-1,j} - E_{i,j}) \,. \end{cases} \tag{1}$$

The score of the best alignment is then $s^*(x,y) = \max_{\pi \in \Pi(x,y)} s(\pi) = \max(0, \max_{i,j}(M_{i,j}))$, and the best-scoring alignment $\pi^*(x,y) = \operatorname{argmax}_{\pi \in \Pi(x,y)} s(\pi)$ can be recovered by backtracking the argmax in the recursion. Both operations can be implemented in $O(nm)$ complexity in time and space; to benefit from modern computational infrastructure, we implemented a version optimized for hardware accelerators such as GPU and TPU, which practically runs in $O(\max(m,n))$ time and in parallel across a batch of sequences to align, using a wavefront algorithm (see Supplementary Methods S1.3). We note that the score of any alignment $\pi \in \Pi(x,y)$ is a linear function of the parameters $\Theta = (S, O, E)$ seen as a $3nm$-dimensional vector, which we write as $s(\pi) = \Theta^\top \Phi(\pi)$ for some mapping $\Phi : \Pi(x,y) \to \mathbb{R}^{3nm}$. This shows that $s^*(x,y)$ is almost everywhere differentiable in Θ, with gradient $\Phi(\pi^*(x,y))$, which potentially allows to backpropagate to Θ any differentiable objective function that depends on the optimal alignment score, such as the one used in the homology detection task. On the other hand, the best alignment $\pi^*(x,y)$ is a piecewise constant function of Θ, and any loss that only depends on $\pi^*(x,y)$, such as how different it is from the correct alignment, can not be backpropagated to the parameters. In the "learning to align" task, however, we need to have a differentiable loss to minimize with respect to Θ when SW finds an alignment $\pi^*(x,y)$ but the correct alignment is $\bar{\pi}$, potentially different from $\pi^*(x,y)$.

To overcome this issue, a first idea is to consider the so-called structured perceptron loss [40], which penalizes the difference between the score of the best alignment and the score of the true alignment: $\ell_{Perceptron}(\Theta) = s(\bar{\pi}) - s^*(x,y)$. Its gradient is simply $\Phi(\bar{\pi}) - \Phi(\pi^*(x,y))$, which can be computed with the SW forward and backward recursions. To create a smoother loss function, we consider two natural extensions to the perceptron loss. The first one is the so-called CRF loss [41] given by $\ell_{CRF}(\Theta) = s(\bar{\pi}) - s^*_\tau(x,y)$ for $\tau \geq 0$. Instead of being the maximum score, $s^*_\tau(x,y) = \tau \log \sum_{\pi \in \Pi(x,y)} e^{s(\pi)/\tau}$ is the soft-maximum score. The CRF loss can also be interpreted as a negative log-likelihood model under a Gibbs distribution, and as shown in [35], its value and gradient can be computed as efficiently as for the perceptron loss, by simply replacing the max operation in the SW recursion (1) by a softmax of the form $\max_\tau(\{u_1, \dots, u_k\}) = \tau \log \sum_{i=1}^k e^{u_k/\tau}$, and adapting the backtracking. The CRF loss thus provides a smooth approximation to the perceptron loss, which converges to it when τ goes to zero. The second extension creates smoothness by random perturbation of Θ, as proposed by [36].

More precisely, we consider a random perturbation $\Theta + \tau Z$ of the parameters Θ, where $\tau \geq 0$ and Z is a standard multivariate normal random vector; then the so-called Fenchel-Young loss [42] associated to the perturbation according to [36] is a valid loss with gradient $\nabla \ell_{FY}(\Theta) = \Phi(\bar{\pi}) - E_Z \Phi(\pi^*(x,y))$. In other words, ℓ_{FY} is a valid loss whose gradient is the expectation of the perceptron loss gradient under random perturbation of the SW parameters. In practice we compute a stochastic gradient (which is enough to optimize the loss function by stochastic gradient descent) by taking the perceptron loss gradient after a single random perturbation of the parameters by τZ. We note, again, that the Fenchel-Young loss is a smooth approximation to the perceptron loss, which converges to it when τ goes to 0.

Homology detection classifier

We treat homology detection as a binary classification problem, aiming to elucidate whether a pair of sequences x and y belong to the same Pfam family or not, given their alignment score $s^*(x, y)$. Since it is well known that the distribution of alignment scores on random sequences strongly depends on their length [43], we must correct for the length effect on the score and pose the following model, inspired by how E-values are defined to assess the significance of alignment scores, for the probability P that x and y are related given their alignment score and lengths n and m:

$$P = \text{Sigmoid}(w_1 s^*(x, y) - w_2 \log(nm) + b), \tag{2}$$

where $w_1, w_2 \in \mathbb{R}_+$ and $b \in \mathbb{R}$ are parameters. These parameters are jointly trained end-to-end alongside the sequence encoder and the differentiable alignment layer using a binary cross-entropy loss. The differentiable alignment layer is thus explicitly trained to output small scores for unrelated sequence pairs, preventing the model from acquiring biases caused by it only being trained on related sequences by the alignment task.

Output head for masked language modelling

We follow [30, 31] and define a masked language modelling task to train the transformer by randomly selecting 15% of the residues in each sequence to be prediction targets. 80% of these are substituted by a special `<MASK>` token, 10% by a randomly chosen residue and the remaining 10% is left unchanged. For a sequence x of length n, we denote by $\mathcal{T} \subseteq [\![1, n]\!]$ the set of target residues, chosen uniformly at random, and by $\tilde{x} \sim \text{perturb}(x; \mathcal{T})$ the resulting randomly perturbed sequence. The goal of the masked language modelling output head is to predict the original value of each perturbed token x_i from the embeddings of the (perturbed) sequence $\phi(\tilde{x})$ as

$$\hat{P}(x_i \mid \tilde{x}) = (\text{Softmax} \circ W^{\text{LM}})(\phi_i(\tilde{x})), \tag{3}$$

where $W^{\text{LM}} : \mathbb{R}^d \to \mathbb{R}^k$ is a learnable, affine mapping that is *not* tied to F in the input embedding layer (see Supplementary Methods S1.2.1). These predicted probabilities $\hat{P}(x_i \mid \tilde{x}) \in \Delta^{k-1}$ are then compared to the ground-truth using a cross-entropy loss,

$$L_{\text{LM}} := \mathbb{E}_x \mathbb{E}_{\mathcal{T}} \mathbb{E}_{\tilde{x}|x,\mathcal{T}} \left[\sum_{i \in \mathcal{T}} - \log \hat{P}(x_i \mid \tilde{x}) \right], \tag{4}$$

where we omitted the dependence of L_{LM} on the parameters ϕ of the sequence encoder and W^{LM}.

Training DEDAL

We first pretrain the sequence encoder on the masked language modelling task for 1 million steps. After that, we jointly train the sequence encoder, differentiable alignment layer, homology detection classifier and masked language modelling output head for 2 million additional steps to minimize a multi-task objective:

$$L_{\text{MT}} := L_\tau + \lambda_{\text{HD}} L_{\text{HD}} + \lambda_{\text{LM}} L_{\text{LM}}, \tag{5}$$

where L_τ is a loss for the alignment task dependent on the relaxation scheme, L_{HD} the binary cross-entropy loss for the homology detection task, L_{LM} the cross-entropy loss for the masked language modelling task and $\lambda_{\text{HD}}, \lambda_{\text{LM}} \in \mathbb{R}_+$ are hyperparameters controlling the relative weight of each loss on the multi-task objective. In both phases, we use the Adam [44] optimizer with a linear warm-up of $8,000$ steps until reaching a maximum learning rate $\text{lr}_{\max}$ that we treat as a hyperparameter, followed by an inverse square root decay schedule.

Metrics

Given two homologous sequences x and y from an evaluation set $\mathcal{D}_{\text{eval}}$ with known correct alignment, let us denote by $m_{\text{true}}(x, y)$ the set of pairs of indices aligned to each other in the correct alignment, and by $m^*(x, y)$ the set of pairs of indices predicted to be aligned by a SW prediction. We assess the performance of the SW prediction in terms of precision, recall and F_1 score defined as:

$$\begin{aligned}
\text{Precision} &:= \frac{\sum_{(x,y,m_{\text{true}}(x,y))\sim\mathcal{D}_{\text{eval}}} |m^*(x,y) \cap m_{\text{true}}(x,y)|}{\sum_{(x,y,m_{\text{true}}(x,y))\sim\mathcal{D}_{\text{eval}}} |m^*(x,y)|}, \\
\text{Recall} &:= \frac{\sum_{(x,y,m_{\text{true}}(x,y))\sim\mathcal{D}_{\text{eval}}} |m^*(x,y) \cap m_{\text{true}}(x,y)|}{\sum_{(x,y,m_{\text{true}}(x,y))\sim\mathcal{D}_{\text{eval}}} |m_{\text{true}}(x,y)|}, \\
\text{F1} &:= 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.
\end{aligned} \tag{6}$$

To evaluate homology detection performance, we use the Area Under the Precision-Recall Curve (AUPRC). We do *not* exclude false positives for sequence pairs that are part of the same Pfam clan yet do not belong to the same Pfam family. Since such sequence pairs might be evolutionarily related, this is a conservative definition of performance that might overestimate the number of erroneous predictions.

To better reflect the relative difficulty of each task, we stratify all alignment and homology detection metrics by Percent IDentity (PID) [45], using the definition of PID from [46], namely,

$$\text{PID} := \frac{\text{IdenticalResidues}}{\text{AlignedResidues} + \text{InternalGaps}}, \tag{7}$$

where the number of identical residues and internal gaps is computed for each pair of related sequences using the ground-truth alignments. For homology detection, only the positive class (sequence pairs belonging to the same Pfam family) is stratified. Negative pairs, for which no ground-truth alignments are available, are shared across all PID bins. Since we chose a version of PID that disregards external gaps, it is largely unaffected by our data augmentation process where we add flanking regions to Pfam domains (Supplementary Figure S6).

In practice, $\mathcal{D}_{\text{eval}}$ consisted of a random subset of $256,000$ held-out sequence pairs for each split (in- and out-of-distribution) and setting (Pfam extended and raw domains).

Baselines

We compared the performance of DEDAL on the alignment and homology detection tasks to three widely-used families of substitution matrices: BLOSUM [32], VTML [33, 34] and PFASUM [20]. For each family, we exhaustively optimized over the following hyperparameters: 1) the matrix number in the family, where we considered four matrix numbers for the BLOSUM family (45, 50, 62, 80), six for the VTML family (10, 20, 40, 80, 160, 200) and, finally, four for the PFASUM family (31, 43, 51, 60); 2) the gap open penalty, taking any integer value between 5 and 16; 3) the gap extent penalty, taking values in $\{0.5, 0.75, 1.0, 1.25, 1.5, 2.0\}$. We selected the best-performing configuration over these $1,008$ combinations as the baseline representing substitution matrices, following the protocol described in Supplementary Methods S1.5.

Data availability

The Uniref50 dataset is freely available under the Creative Commons Attribution (CC BY 4.0) License from `https://www.uniprot.org`. Pfam is freely available under the Creative Commons Zero ("CC0") licence from `https://pfam.xfam.org`

Code availability

The code used in this study is freely available under an Apache 2.0 license at `https://github.com/google-research/google-research/tree/master/dedal`

References

1. Prakash, T. & Taylor, T. D. Functional assignment of metagenomic data: challenges and applications. *Brief Bioinform.* **13** (2012).
2. Lockless, S. W. & Ranganathan, R. Evolutionarily Conserved Pathways of Energetic Connectivity in Protein Families. *Science* **286,** 295–299 (1999).
3. Marks, D. S. *et al.* Protein 3D structure computed from evolutionary sequence variation. *PloS one* **6,** e28766 (2011).
4. Jumper, J. *et al.* Highly accurate protein structure prediction with AlphaFold. *Nature* **596,** 583–589 (2021).
5. Smith, T. F., Waterman, M. S., *et al.* Identification of common molecular subsequences. *J. Mol. Biol.* **147,** 195–197 (1981).
6. Altschul, S., W. Gish, W. Miller, Myers, E. & Lipman, D. Basic local alignment search tools. *J. Mol. Bol.* **215,** 403–410 (1990).
7. Pearson, W. R. Rapid and sensitive sequence comparisons with FASTP and FASTA. *Meth. Enzymol.* **183,** 63–98 (1990).
8. Altschul, S. *et al.* Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res.* **25,** 3389–3402 (1997).
9. Landan, G. & Graur, D. Characterization of pairwise and multiple sequence alignment errors. *Gene* **441,** 141–147 (2009).
10. Lobb, B., Kurtz, D. A., Moreno-Hagelsieb, G. & Doxey, A. C. Remote homology and the functions of metagenomic dark matter. *Front Genet.* **6** (2015).
11. Yu, C.-N. J., Joachims, T., Elber, R. & Pillardy, J. Support vector training of protein alignment models. *J. Comput. Biol.* **15,** 867–880 (2008).
12. Fitch, W. M. & Smith, T. F. Optimal sequence alignments. *Proc. Natl. Acad. Sci. U.S.A.* **80,** 1382–1386 (1983).
13. Waterman, M. S., Eggert, M. & Lander, E. Parametric sequence comparisons. *Proc. Natl. Acad. Sci. U.S.A.* **89,** 6090–6093 (1992).
14. Gusfield, D., Balasubramanian, K. & Naor, D. Parametric optimization of sequence alignment. *Algorithmica* **12,** 312–326 (1994).
15. Waterman, M. S. Parametric and ensemble sequence alignment algorithms. *Bull. Math. Biol.* **56,** 743–767 (1994).
16. Vingron, M. & Waterman, M. S. Sequence alignment and penalty choice. Review of concepts, case studies and implications. *J. Mol. Biol.* **235,** 1–12 (1994).
17. Gusfield, D. & Stelling, P. Parametric and inverse-parametric sequence alignment with XPARAL. *Methods Enzymol.* **266,** 481–494 (1996).
18. Pachter, L. & Sturmfels, B. Parametric inference for biological sequence analysis. *Proc. Natl. Acad. Sci. U.S.A.* **101,** 16138–16143 (2004).
19. Henikoff, S. & Henikoff, J. G. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. U.S.A.* **89,** 10915–10919 (1992).

20. Keul, F., Hess, M., Goesele, M. & Hamacher, K. PFASUM: a substitution matrix from Pfam structural alignments. *BMC Bioinform.* **18,** 1–14 (2017).

21. Sun, F., Fernández-Baca, D. & Yu, W. *Inverse Parametric Sequence Alignment* in *Computing and Combinatorics* (eds Ibarra, O. H. & Zhang, L.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2002), 97–106.

22. Saigo, H., Vert, J.-P. & Akutsu, T. Optimizing amino acid substitution matrices with a local alignment kernel. *BMC Bioinform.* **7,** 246 (May 2006).

23. Kececioglu, J. & Kim, E. *Simple and Fast Inverse Alignment* in *Research in Computational Molecular Biology* (eds Apostolico, A., Guerra, C., Istrail, S., Pevzner, P. A. & Waterman, M.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2006), 441–455.

24. Bepler, T. & Berger, B. Learning protein sequence embeddings using information from structure. *arXiv preprint arXiv:1902.08661* (2019).

25. Morton, J. T. *et al.* Protein Structural Alignments From Sequence. *bioRxiv preprint 2020.11.03.365932* (2020).

26. Petti, S. *et al.* End-to-end learning of multiple sequence alignments with differentiable Smith-Waterman. *bioRxiv preprint 2021.10.23.465204* (2021).

27. Vaswani, A. *et al. Attention is all you need* in *Advances in neural information processing systems 30* (eds Guyon, I. *et al.*) (Curran Associates, Inc., 2017), 5998–6008.

28. Suzek, B. E. *et al.* UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics* **31,** 926–932 (2015).

29. Mistry, J. *et al.* Pfam: The protein families database in 2021. *Nucleic Acids Res.* **49,** D412–D419 (2021).

30. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)* (eds Burstein, J., Doran, C. & Solorio, T.) (Association for Computational Linguistics, 2019), 4171–4186.

31. Rives, A. *et al.* Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proc. Natl. Acad. Sci. U.S.A.* **118,** e2016239118 (2021).

32. Henikoff, S. & Henikoff, J. G. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. U.S.A.* **89,** 10915–10919 (1992).

33. Müller, T. & Vingron, M. Modeling amino acid replacement. *J. Comput. Biol.* **7,** 761–776 (2000).

34. Müller, T., Spang, R. & Vingron, M. Estimating amino acid substitution models: a comparison of Dayhoff's estimator, the resolvent approach and a maximum likelihood method. *Mol. Biol. Evol.* **19,** 8–13 (2002).

35. Mensch, A. & Blondel, M. *Differentiable dynamic programming for structured prediction and attention* in *Proceedings of the 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) **80** (PMLR, 2018), 3462–3471.

36. Berthet, Q. *et al. Learning with differentiable perturbed optimizers* in *Advances in neural information processing systems 33* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H.) (2020).

37. Rao, R. *et al.* Evaluating protein transfer learning with TAPE. *Advances in neural information processing systems* **32,** 9689 (2019).

38. The UniProt Consortium. UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Res.* **49,** D480–D489 (2021).

39. Steinegger, M. & Söding, J. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nat. Biotechnol.* **35,** 1026–1028 (2017).

40. Collins, M. *Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms* in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10* (Association for Computational Linguistics, 2002), 1–8.

41. Lafferty, J., McCallum, A. & Pereira, F. *Conditional random fields: Probabilistic models for segmenting and labeling sequence data* in *Proc. 18th International Conf. on Machine Learning* (Morgan Kaufmann, San Francisco, CA, 2001), 282–289.

42. Blondel, M., Martins, A. F. & Niculae, V. Learning with Fenchel-Young losses. *J. Mach. Learn. Res.* **21,** 1–69 (2020).

43. Karlin, S. & Altschul, S. F. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci. U.S.A.* **87,** 2264–2268 (1990).

44. Kingma, D. P. & Ba, J. *Adam: A method for stochastic optimization* in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (eds Bengio, Y. & LeCun, Y.) (2015).

45. Raghava, G. P. & Barton, G. J. Quantification of the variation in percentage identity for protein sequence alignments. *BMC Bioinform.* **7,** 1–4 (2006).

46. Doolittle, R. F. Similar amino acid sequences: chance or common ancestry? *Science* **214,** 149–159 (1981).

Author contributions

F.L.L. contributed to the development of the method, implemented most of the code, designed and ran most experiments, drafted the manuscript. Q.B. contributed to the development of the method, to the code, and to the experiments. M.B. contributed to the development of the method and to the code. O.T. designed and implemented major parts of the code and contributed to the experiments. J.P.V. designed the initial project, contributed to the method development and drafted the manuscript. All authors provided regular feedback to all aspects of the work, reviewed code and contributed to the writing of the final manuscript.

Supplementary Material to: Deep embedding and alignment of protein sequences

Felipe Llinares-López, Quentin Berthet, Mathieu Blondel,
Olivier Teboul and Jean-Philippe Vert*

Google Research, Brain team

November 15, 2021

Contents

S1 Supplementary Methods **2**
S1.1 Pfam domain sequence extension 2
S1.2 DEDAL model and training 3
S1.2.1 Contextual sequence embeddings 3
S1.2.2 Parameterizer 4
S1.2.3 Architecture and hyperparameters. 5
S1.3 Accelerator-friendly implementation of the (smooth) SW algorithm. 5
S1.4 Efficiently differentiating the soft-maximum SW score 6
S1.5 Selecting the best-performing substitution matrix baseline 8

S2 Supplementary Results **10**
S2.1 Ablation study 10
S2.2 Results on the TAPE benchmark 12

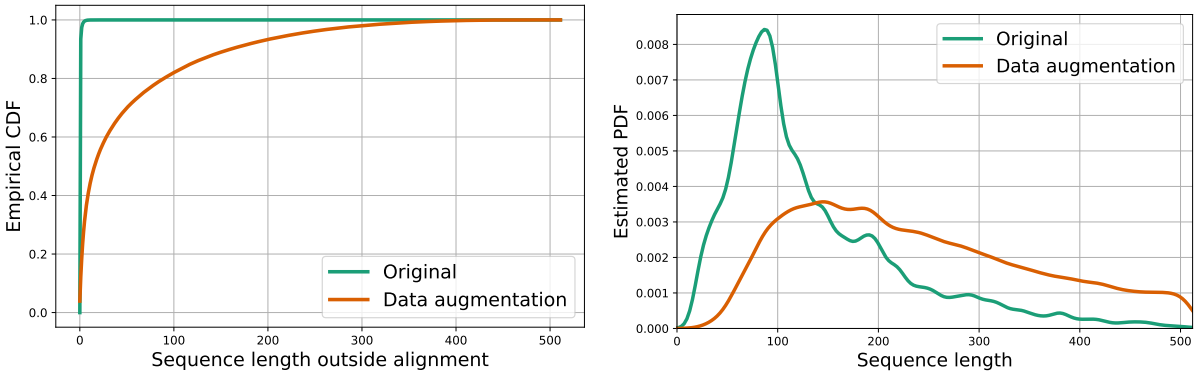
S3 Supplementary Figures **14**

*Correspondance: jpvert@google.com

S1 Supplementary Methods

S1.1 Pfam domain sequence extension

As explained in the main Methods, we use pairs of aligned domain sequences from Pfam-A seed as ground-truth alignments, to supervise the "learning to align" task of DEDAL and assess the performance of various alignment algorithms. A drawback of using Pfam-A seed alignments as targets, however, stems from the fact that these are *de facto* global alignments. This is illustrated in Figure S1a, where we see that the vast majority of pairwise alignments from the original Pfam-A seed dataset have no flanking sequence tails outside the alignment region. Thus, models naively trained using these alignments as supervision targets will be heavily biased towards global alignment, which might impede their ability to predict local alignments whenever necessary.



(a) (b)

Figure S1: (a) The empirical CDF of the number of residues not mapping to matches or internal gaps in a target alignment. (b) A kernel density estimate of the sequence length distribution. A random set of 128,000 sequence pairs from the training set was used for both figures.

To overcome this limitation, we apply *data augmentation* to the Pfam-A seed dataset, creating *local* pairwise sequence alignment targets as follows. Firstly, we cross-reference each Pfam-A seed sequence with the UniProtKB entry in which it occurs. This provides a ground-truth *context* from which to incorporate biologically relevant flanking tails, given by the L_{prefix} and L_{suffix} residues that precede and follow the Pfam-A seed sequence in its UniProtKB context, respectively.[1]. To further increase data diversity, L_{prefix} and L_{suffix} are chosen independently at random every time a Pfam-A seed sequence is drawn as part of a sequence pair for training or evaluation. In particular, let L_{seq} be the length of a given Pfam-A seed sequence and $L_{\text{ctx}} = L_{\text{prefix,max}} + L_{\text{seq}} + L_{\text{suffix,max}}$ the length of the UniProtKB sequence in which it occurs, where $L_{\text{prefix,max}}$ and $L_{\text{suffix,max}}$ are the total number of residues available as prefix and suffix, respectively. We sample L_{prefix} and L_{suffix} as:

$$\begin{aligned} L_{\text{tails}} &\sim U(0, \min(L_{\text{ctx}}, L_{\text{model}}) - L_{\text{seq}}), \\ L_{\text{prefix}} &\sim U(\max(0, L_{\text{tails}} - L_{\text{suffix,max}}), \min(L_{\text{tails}}, L_{\text{prefix,max}})), \\ L_{\text{suffix}} &= L_{\text{tails}} - L_{\text{prefix}}, \end{aligned} \tag{1}$$

with $L_{\text{model}} = 512$ being the maximum sequence length we consider for this task. In other words, the total length of flanking tails being added (prefix *and* suffix) is sampled uniformly between zero and the maximum possible, given the context length and the computational budget. These L_{tails} residues are subsequently allocated to prefix and suffix uniformly at random among all

[1]For the sake of simplicity, we discard 550 Pfam-A seed sequences that occur more than once within their corresponding UniProtKB entry.

valid choices. A high-level overview of the data augmentation process is depicted in Figure S2. As shown in Figure S1a, this data augmentation process greatly increases the diversity of alignment targets relative to the original data. Moreover, the augmented input sequences have a significantly less skewed length distribution, as can be seen in Figure S1b.

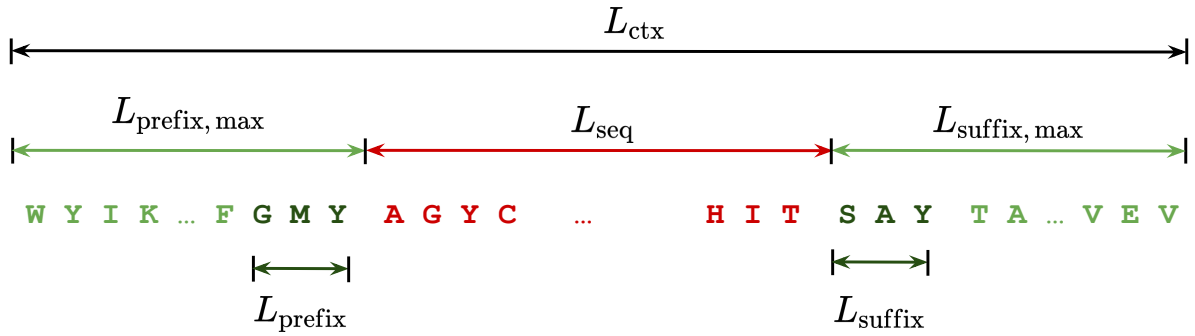


Figure S2: Conceptual illustration of the data augmentation process. A Pfam-A seed sequence (red) is shown alongside the remainder of the UniProtKB sequence it is a part of (green). Each time the Pfam-A seed sequence is drawn in a sequence pair, a random amount of prefix and suffix is added to simulate a local alignment target (dark green).

S1.2 DEDAL model and training

In this section we provide additional details on the DEDAL model, and how it is trained. DEDAL contains a *sequence encoder* that computes continuous representations of protein sequences and multiple task-specific *output heads* that make predictions based on these continuous representations. All model components are jointly trained end-to-end using a multi-task loss, with the exception of the output heads for downstream tasks, which are fine-tuned independently for each task. The remainder of this section introduces each model component in turn, concluding with a description of the training process.

S1.2.1 Contextual sequence embeddings

The sequence encoder is a parametric function that maps sequences $x = (x_1, x_2, \ldots, x_n)$ over a finite alphabet ($x_i \in \mathcal{A}$ for all $i = 1, \ldots, n$) to a sequence of d-dimensional vectors of the same length $T_\phi(x) = (T_{\phi,1}(x), T_{\phi,2}(x), \ldots, T_{\phi,n}(x))$. Crucially, its architecture was chosen so that the *embedding* for the i-th input residue $T_{\phi,i}(x) \in \mathbb{R}^d$ depends on the entire sequence x rather than on x_i alone, making these sequence embeddings *contextual*. To this end, we use a transformer encoder [1].Transformer-based models achieve state-of-the-art performance across a wide range of problems in natural language processing and, more recently, computer vision. Unlike convolutional or recurrent architectures, transformers make extensive use of *attention* to model long-range interactions between tokens directly, thus excelling at capturing non-local context.

Tokenization. In our case, the alphabet $\mathcal{A} = [\![1, k]\!]$ consists of $k = 27$ *tokens* corresponding to (i) the 20 standard amino acids, (ii) the less common Pyrrolysine (O) and Selenocysteine (U) amino acids, (iii) the "ambiguous" amino acid codes B, Z and X and (iv) the special tokens `<EOS>`, which we append to each sequence, and `<MASK>`, which is used for masked language modelling only.

Input embeddings. Similar to [1, 2], the first layer of our sequence encoder is an *input embedding layer* that additively encodes residue identity and position information for each

token in the input sequence. Given a sequence $x = (x_1, x_2, \ldots, x_n) \in \mathcal{A}^n$, we define $X \in \mathbb{R}^{n \times k}$ to be the *one-hot* representation of x, i.e., $X_{i,j} = \delta_j(x_i)$. The input embedding layer computes non-contextual, continuous representations for each residue as

$$Z_0 = XF + P, \tag{2}$$

where $F \in \mathbb{R}^{k \times d}$ contains learnable input embeddings for each token in $\mathcal{A}$ and $P \in \mathbb{R}^{n \times d}$ is a matrix of sinusoidal, absolute position embeddings [1].

Transformer encoder layers. The non-contextual embeddings $Z_0 \in \mathbb{R}^{n \times k}$ are then fed to a stack of L *transformer encoder layers* which use *multi-headed self-attention* followed by a position-wise multilayer perceptron (*MLP*). Together, these adaptively incorporate into each residue's embedding information about the other tokens in the sequence. In particular, we use transformer encoder layers with pre-layer normalization [3, 4],

$$\begin{aligned} \widetilde{Z}_l &= Z_{l-1} + (\text{Dropout} \circ \text{MultiHeadedSelfAttention} \circ \text{LayerNorm})(Z_{l-1}), \\ Z_l &= \widetilde{Z}_l + (\text{Dropout} \circ \text{MLP} \circ \text{LayerNorm})(\widetilde{Z}_l), \end{aligned} \tag{3}$$

where $\circ$ denotes function composition.

The multi-headed self-attention mechanism combines the output from h independent self-attention layers as

$$\begin{aligned} \text{MultiHeadedSelfAttention}(Z) &= \sum_{i=1}^{h} (W_i^O \circ \text{SelfAttention}_i)(Z), \\ \text{SelfAttention}_i(Z) &= \text{Dropout}\left(\text{Softmax}\left(\frac{W_i^Q(Z)(W_i^K(Z))^T}{\sqrt{d/h}}\right)\right) W_i^V(Z), \end{aligned} \tag{4}$$

where $W_i^Q, W_i^Q, W_i^V : \mathbb{R}^d \to \mathbb{R}^{d/h}$ and $W_i^O : \mathbb{R}^{d/h} \to \mathbb{R}^d$ are all learnable, affine mappings. These and the softmax non-linearity are applied row-wise to their respective $n \times d$, $n \times d/h$ and $n \times n$ inputs.

The MLP contains a single hidden layer that uses a Gaussian Error Linear Unit (GELU) [5] element-wise non-linearity, namely,

$$\text{MLP}(Z) = (W^{\text{out}} \circ \text{Dropout} \circ \text{GELU} \circ W^{\text{in}})(Z), \tag{5}$$

where $W^{\text{in}} : \mathbb{R}^d \to \mathbb{R}^{4d}$ and $W^{\text{out}} : \mathbb{R}^{4d} \to \mathbb{R}^d$ are also learnable, affine mappings applied row-wise to their inputs.

Output normalization. Finally, layer normalization is applied to the output of the L-th transformer encoder layer to obtain the contextual embeddings $T_\phi(x) \in \mathbb{R}^{n \times d}$ for an input sequence $x \in \mathcal{A}^n$,

$$T_\phi(x) := \text{LayerNorm}(Z_L). \tag{6}$$

S1.2.2 Parameterizer

Once each sequence is mapped to a continuous representation, we compute matrices of parameters to be used by the SW alignment algorithm from the continuous representations. This allows the SW algorithm to rely on information other than the identity of individual residue pairs by modelling the substitution scores, gap open and gap extend penalties as a parametric

function of the (contextual) embeddings $T_\phi(x)$ and $T_\phi(y)$ of the sequences to be aligned. In practice we implement these as symmetric bilinear forms,

$$\begin{aligned} S_{i,j}(x,y) &:= T^T_{\phi,i}(x) W^S T_{\phi,j}(y) + b^S, \\ O_{i,j}(x,y) &:= \text{Softplus}\left(T^T_{\phi,i}(x) W^O T_{\phi,j}(y) + b^O\right), \\ E_{i,j}(x,y) &:= \text{Softplus}\left(T^T_{\phi,i}(x) W^E T_{\phi,j}(y) + b^E\right), \end{aligned} \tag{7}$$

where $W^S, W^O, W^E \in \mathbb{R}^{d\times d}$ are symmetric matrices and $b^S, b^O, b^E \in \mathbb{R}$ scalar biases, respectively. Together, these form the set of parameters β of the parameterizer function P_β. An element-wise softplus non-linearity is used to ensure gap penalties remain non-negative.

This model is a strict generalization of the traditional parameterization of the SW algorithm based on substitution matrices. We can recover these as a particular case by (i) using a one-hot representation as embeddings, i.e., $T_\phi(x) = X$ and $T_\phi(y) = Y$, (ii) restricting the gap penalties to be position-independent, i.e., $O_{i,j} := O$, $E_{i,j} := E$ with $O, E \in \mathbb{R}_+$ and (iii) fixing $W^S \in \mathbb{R}^{k\times k}$ to the desired substitution matrix. In contrast, we jointly train the transformer-based sequence encoder and the alignment layer, resulting in a substantially more expressive parameterization.

S1.2.3 Architecture and hyperparameters.

We adopt the architecture of the smallest transformer-based model in [2], which uses $L = 6$ transformer encoder layers with $h = 12$ heads per layer and embeddings of dimension $d = 768$. Glorot initialization [6] is used for all weight matrices with the sole exception of the input embeddings table, which are initialized from a standard normal distribution. The dropout rate is set to 0.1 throughout. A global batch size of 128 sequences or, when applicable, sequence pairs, was used in all experiments. No attempts were made to optimize over these hyperparameters. Additionally, DEDAL was trained with maximum learning rate $\text{lr}_{\max} = 10^{-4}$ and multi-task loss weights $\lambda_{\text{HD}} = 20$, $\lambda_{\text{LM}} = \lambda_{\text{HD}}/2$. Unless stated otherwise (e.g. ablations), DEDAL used the perturbation-based approach in [7] with relaxation strength $\tau = 0.1$ to smooth the SW alignments during training. For computational considerations, these hyperparameters were selected after a lightweight exploration of model performance on the in-distribution and out-of-distribution validation splits, which have no overlap with the held-out test set.

S1.3 Accelerator-friendly implementation of the (smooth) SW algorithm.

As explained in the Methods section, we use the SW algorithm to find the maximum scoring alignment (or a soft maximum in the case of the CRF loss) with the dynamic programming recursion over the $(n+1)\times(m+1)$ matrices M, X and Y defined for any $i = 1,\ldots,n$ and $j = 1,\ldots,m$ by $M_{i,0} = M_{0,j} = X_{i,0} = X_{0,j} = Y_{i,0} = Y_{0,j} = -\infty$ and:

$$\begin{cases} M_{i,j} &= S_{i,j} + \max_\tau(0, M_{i-1,j-1}, X_{i-1,j-1}, Y_{i-1,j-1}), \\ X_{i,j} &= \max_\tau(M_{i,j-1} - O_{i,j}, X_{i,j-1} - E_{i,j}), \\ Y_{i,j} &= \max_\tau(M_{i-1,j} - O_{i,j}, X_{i-1,j} - O_{i,j}, Y_{i-1,j} - E_{i,j}), \end{cases} \tag{8}$$

where $\max_\tau(\{u_1,\ldots,u_k\}) := \tau \log \sum_{i=1}^k e^{u_k/\tau}$ is the soft-max when $\tau > 0$, or the max operator when $\tau = 0$. The (soft) best-scoring alignment needed to compute the gradient of the score and of the different alignment loss functions with respect to S, O and E is then computed by appropriate backtracking.

The widespread availability of modern hardware accelerators such as Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs) has been partly responsible for the recent success of deep learning models. These hardware resources excel at vectorized computation, perhaps most notably matrix-matrix and matrix-vector multiplication, which are the fundamental building blocks of most commonly-used deep learning layers. In contrast, the dynamic

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	$\pm\infty$	$\pm\infty$	$\pm\infty$	$\pm\infty$
$\pm\infty$	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	$\pm\infty$	$\pm\infty$	$\pm\infty$
$\pm\infty$	$\pm\infty$	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	$\pm\infty$	$\pm\infty$
$\pm\infty$	$\pm\infty$	$\pm\infty$	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	$\pm\infty$
$\pm\infty$	$\pm\infty$	$\pm\infty$	$\pm\infty$	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8

Table S1: Effect of applying the mapping $f^{(+)}$ (resp. $f^{(-)}$) to a tensor $A \in \mathbb{R}^{5\times 8}$, representing SW parameters for a pair of sequences of lengths $n = 5$ and $m = 9$. Entries in the output tensor not corresponding to any entry of A are padded with $+\infty$ (resp. $-\infty$).

program underlying the SW algorithm appears to be inherently sequential. Arguably, its most natural description involves firstly iteratively computing $M_{i,j}$, $X_{i,j}$ and $Y_{i,j}$ using a double `for` loop along the lengths of sequences x and y, followed by backtracking. Unfortunately, such an implementation would only profit from vectorization along the batch axis, i.e., carrying out the dynamic program updates for multiple sequence pairs simultaneously, remaining otherwise purely sequential along both length axes i and j. To prevent the differentiable alignment layer from becoming a computational bottleneck, we implement a wavefront transformation that allows expressing the SW algorithm in terms of a single `for` loop, leading to a computational footprint comparable to that of Recurrent Neural Networks (RNNs). Most importantly, we aim to accomplish this using solely operations supported off-the-shelf by frameworks such as TensorFlow [8], JAX [9] or PyTorch [10] for maximum compatibility and ease of use.

Suppose $S, D, O \in \mathbb{R}^{b\times n\times m}$ are tensors representing the (contextual) substitution scores, gap open and gap extend penalties of a batch of b sequence pairs of lengths at most n and m, respectively. Entries corresponding to sequences strictly shorter than n (resp. m) are assumed to have been padded with $-\infty$ for S and $+\infty$ for D and E. We also assume w.l.o.g. that $n \leq m$. In order to vectorize the SW algorithm, we exploit that the dynamic program updates can be computed in parallel for all entries (i, j) along the same anti-diagonal, i.e., $i + j = k$ for some $k \in [\![1, n + m - 1]\!]$. To reduce the number of non-contiguous indexing operations, we define mappings $f^{(+)}, f^{(-)} : \mathbb{R}^{b\times n\times m} \to \mathbb{R}^{b\times n\times n+m-1}$ to rearrange the inputs S, D and E mapping their anti-diagonals to a single axis in the output tensors. Namely,

$$f^{(+)}_{s,i,k}(A) = \begin{cases} A_{s,i,k-i-1}, & \text{if} \quad k-i-1 \in [\![1, m]\!], \\ +\infty, & \text{otherwise}, \end{cases} \tag{9}$$

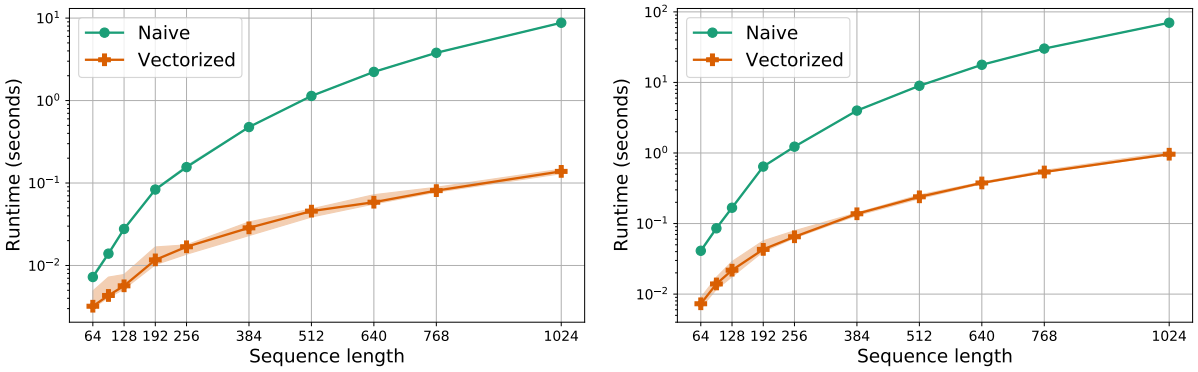
with $f^{(-)}$ defined analogously but using $-\infty$ as a sentinel value for invalid entries. A conceptual illustration of this transformation is shown in Figure S1. Setting $\widetilde{S} := f^{(-)}(S)$, $\widetilde{D} := f^{(+)}(D)$ and $\widetilde{E} := f^{(+)}(E)$, we can rewrite the dynamic program updates in Equation 8 as

$$\begin{aligned} M_{i,k} &= \max\nolimits_\tau \left(0, M_{i-1,k-2}, X_{i-1,k-2}, Y_{i-1,k-2}\right) + \widetilde{S}_{i,k}, \\ X_{i,k} &= \max\nolimits_\tau \left(M_{i-1,k-1} - \widetilde{D}_{i,k}, X_{i-1,k-1} - \widetilde{E}_{i,k}, Y_{i-1,k-1} - D_{i,j}\right), \\ Y_{i,k} &= \max\nolimits_\tau \left(M_{i,k-1} - \widetilde{D}_{i,k}, Y_{i,k-1} - \widetilde{E}_{i,k}\right), \end{aligned} \tag{10}$$

which can be trivially vectorized along the batch and i axes, requiring only a single `for` loop on $k \in [\![1, n + m - 1]\!]$. Despite the fact that this change does not alter the computational complexity of the algorithm, we found reducing the amount of strictly sequential computation to have a dramatic impact on runtime in practice, as shown in Table S3.

S1.4 Efficiently differentiating the soft-maximum SW score

The gradient of the soft-maximum score $s^*_\tau(x, y) = \tau \log \sum_{\pi\in\Pi(x,y)} e^{s(\pi)/\tau}$ can in principle be computed using modern automatic differentiation (AD) software [8–10]. However, we have



(a) (b)

Figure S3: Runtime to align a batch of sequence pairs as a function of the length of the sequences for the proposed vectorization approach and a naive baseline implemented as a double `for` loop. Times shown reflect the forward pass only, computed in inference mode ($\tau = 0$) using 4 TPU v3 chips. All data points were obtained as the median of 10 trials, with error bars reflecting the minimum and maximum runtimes among these. (a) Batch size fixed to 128 sequence pairs. (b) Batch size fixed to 1024 sequence pairs.

found using a custom implementation of $\nabla s^*_\tau(x, y)$ advantageous occasionally, particularly in terms of memory usage. For the sake of completeness, we include a detailed derivation of this custom implementation in this section.

Suppose that the recursive updates described in Equation 8 have been run until termination and define $T := \max_\tau \left(0, \max_{i,j}(M_{i,j})\right)$ such that $s^*_\tau(x, y) = T$. Our derivation will proceed in two steps. First, we express the sought-after partial derivatives $\frac{\partial s^*_\tau(x,y)}{\partial S_{i,j}}$, $\frac{\partial s^*_\tau(x,y)}{\partial O_{i,j}}$ and $\frac{\partial s^*_\tau(x,y)}{\partial E_{i,j}}$ as a function of the *adjoint operators* $\bar{M}_{i,j} := \frac{\partial s^*_\tau(x,y)}{\partial M_{i,j}}$, $\bar{X}_{i,j} := \frac{\partial s^*_\tau(x,y)}{\partial X_{i,j}}$, $\bar{Y}_{i,j} := \frac{\partial s^*_\tau(x,y)}{\partial Y_{i,j}}$ and of the (local) partial derivatives of the update rules in Equation 8. Next, we derive update rules to iteratively evaluate all of these adjoint operators.

Using the chain rule, we may write

$$\begin{aligned} \frac{\partial s^*_\tau(x, y)}{\partial S_{i,j}} &= \frac{\partial s^*_\tau(x, y)}{\partial M_{i,j}} \frac{\partial M_{i,j}}{\partial S_{i,j}} \\ &= \bar{M}_{i,j}, \end{aligned} \tag{11}$$

for the substitution scores. Similarly, for the gap open penalties we have

$$\begin{aligned} \frac{\partial s^*_\tau(x, y)}{\partial O_{i,j}} &= \frac{\partial s^*_\tau(x, y)}{\partial X_{i,j}} \frac{\partial X_{i,j}}{\partial O_{i,j}} + \frac{\partial s^*_\tau(x, y)}{\partial Y_{i,j}} \frac{\partial Y_{i,j}}{\partial O_{i,j}} \\ &= \bar{X}_{i,j} \left(-\frac{\partial X_{i,j}}{\partial M_{i,j-1}}\right) + \bar{Y}_{i,j} \left(-\frac{\partial Y_{i,j}}{\partial M_{i-1,j}} - \frac{\partial Y_{i,j}}{\partial X_{i-1,j}}\right), \end{aligned} \tag{12}$$

where we used that

$$\frac{\partial X_{i,j}}{\partial O_{i,j}} = \frac{\partial \max_\tau(\{M_{i,j-1} - O_{i,j}, X_{i,j-1} - E_{i,j}\})}{\partial O_{i,j}} = -\frac{\partial X_{i,j}}{\partial M_{i,j-1}} \tag{13}$$

and that

$$\frac{\partial Y_{i,j}}{\partial O_{i,j}} = \frac{\partial \max_\tau(\{M_{i-1,j} - O_{i,j}, X_{i-1,j} - O_{i,j}, Y_{i-1,j} - E_{i,j}\})}{\partial O_{i,j}} = -\frac{\partial Y_{i,j}}{\partial M_{i-1,j}} - \frac{\partial Y_{i,j}}{\partial X_{i-1,j}} \tag{14}$$

in the last step of Equation 12. Based on the same approach, the partial derivatives with respect to the gap extend penalties are given by

$$\begin{aligned}\frac{\partial s^*_\tau(x,y)}{\partial E_{i,j}} &= \frac{\partial s^*_\tau(x,y)}{\partial X_{i,j}}\frac{\partial X_{i,j}}{\partial E_{i,j}} + \frac{\partial s^*_\tau(x,y)}{\partial Y_{i,j}}\frac{\partial Y_{i,j}}{\partial E_{i,j}} \\ &= \bar{X}_{i,j}\left(-\frac{\partial X_{i,j}}{\partial X_{i,j-1}}\right) + \bar{Y}_{i,j}\left(\frac{\partial Y_{i,j}}{\partial Y_{i-1,j}}\right),\end{aligned} \tag{15}$$

where again we used that $\frac{\partial X_{i,j}}{\partial E_{i,j}} = -\frac{\partial X_{i,j}}{\partial X_{i,j-1}}$ and $\frac{\partial Y_{i,j}}{\partial E_{i,j}} = -\frac{\partial Y_{i,j}}{\partial Y_{i-1,j}}$, which can be shown analogously to Equations 13 and 14.

To compute the adjoint operators, we use a recursive implementation with a computation flow analogous to backtracking for the SW algorithm albeit with different update rules. Namely,

$$\begin{aligned}\bar{M}_{i,j} &= \frac{\partial s^*_\tau(x,y)}{\partial T}\frac{\partial T}{\partial M_{i,j}} + \frac{\partial s^*_\tau(x,y)}{\partial M_{i+1,j+1}}\frac{\partial M_{i+1,j+1}}{\partial M_{i,j}} + \frac{\partial s^*_\tau(x,y)}{\partial X_{i,j+1}}\frac{\partial X_{i,j+1}}{\partial M_{i,j}} + \frac{\partial s^*_\tau(x,y)}{\partial Y_{i+1,j}}\frac{\partial Y_{i+1,j}}{\partial M_{i,j}} \\ &= \frac{\partial T}{\partial M_{i,j}} + \bar{M}_{i+1,j+1}\frac{\partial M_{i+1,j+1}}{\partial M_{i,j}} + \bar{X}_{i,j+1}\frac{\partial X_{i,j+1}}{\partial M_{i,j}} + \bar{Y}_{i+1,j}\frac{\partial Y_{i+1,j}}{\partial M_{i,j}}, \\ \bar{X}_{i,j} &= \frac{\partial s^*_\tau(x,y)}{\partial M_{i+1,j+1}}\frac{\partial M_{i+1,j+1}}{\partial X_{i,j}} + \frac{\partial s^*_\tau(x,y)}{\partial X_{i,j+1}}\frac{\partial X_{i,j+1}}{\partial X_{i,j}} + \frac{\partial s^*_\tau(x,y)}{\partial Y_{i+1,j}}\frac{\partial Y_{i+1,j}}{\partial X_{i,j}} \\ &= \bar{M}_{i+1,j+1}\frac{\partial M_{i+1,j+1}}{\partial X_{i,j}} + \bar{X}_{i,j+1}\frac{\partial X_{i,j+1}}{\partial X_{i,j}} + \bar{Y}_{i+1,j}\frac{\partial Y_{i+1,j}}{\partial X_{i,j}}, \\ \bar{Y}_{i,j} &= \frac{\partial s^*_\tau(x,y)}{\partial M_{i+1,j+1}}\frac{\partial M_{i+1,j+1}}{\partial Y_{i,j}} + \frac{\partial s^*_\tau(x,y)}{\partial Y_{i+1,j}}\frac{\partial Y_{i+1,j}}{\partial Y_{i,j}} \\ &= \bar{M}_{i+1,j+1}\frac{\partial M_{i+1,j+1}}{\partial Y_{i,j}} + \bar{Y}_{i+1,j}\frac{\partial Y_{i+1,j}}{\partial Y_{i,j}}.\end{aligned} \tag{16}$$

The (local) partial derivatives of the update rules in Equation 8 can all be trivially evaluated, as they are all instances of the smoothed max operator $\max_\tau(\{u_1,\ldots,u_k\}) = \tau \log \sum_{i=1}^k e^{u_k/\tau}$ whose partial derivatives $\frac{\partial \max_\tau(\{u_1,\ldots,u_k\})}{\partial u_i} = \exp\left(\frac{u_i - \max_\tau(\{u_1,\ldots,u_k\})}{\tau}\right)$ are given by the well-known *softmax* function. For instance, one may write $\frac{\partial T}{\partial M_{i,j}} = \exp\left(\frac{M_{i,j}-T}{\tau}\right)$, $\frac{\partial M_{i+1,j+1}}{\partial M_{i,j}} = \exp\left(\frac{M_{i,j}-M_{i+1,j+1}}{\tau}\right)$ and $\frac{\partial Y_{i+1,j}}{\partial X_{i,j}} = \exp\left(\frac{X_{i,j}+O_{i,j}-Y_{i+1,j}}{\tau}\right)$. Lastly, we highlight that a careful implementation can reuse the memory used to store the values of $M_{i,j}$, $X_{i,j}$ and $Y_{i,j}$ during the forward pass to also store $\bar{M}_{i,j}$, $\bar{X}_{i,j}$ and $\bar{Y}_{i,j}$ during the backward pass, roughly halving memory usage relative to a brute-force implementation.

S1.5 Selecting the best-performing substitution matrix baseline

We use a combination of in-distribution and out-of-distribution performance on the alignment and homology detection tasks to optimize the matrix number, gap open and gap extend penalties for each matrix family. This criterion preferentially selects versatile models that perform reliably well across all tasks under consideration over models that excel at a single task. Supposing there are N metrics[2] we wish to take into account and K different hyperparameter settings to select from, we define an additive, self-normalized scalar summary of these N metrics as

$$f_k := \frac{1}{N}\sum_{i=1}^{N}\frac{f_{i,k} - \min_k f_{i,k}}{\max_k f_{i,k} - \min_k f_{i,k}}, \tag{17}$$

[2]For the sake of simplicity, we tacitly assume all metrics have been defined so that larger values imply better performance. In practice, we multiply the metric by minus one before normalization and aggregation whenever this is not the case.

where $f_{i,k}$ is the value that the k-th hyperparameter setting attains for the i-th metric. The optimal hyperparameters are then simply determined as $k^* = \text{argmax}_k f_k$. Concretely, in this work we use $N = 4$ metrics, namely, (i) in-distribution alignment F1 score, (ii) out-of-distribution alignment F1 score, (iii) in-distribution homology detection AUPRC for remote homologs (PID < 0.1) and (iv) out-of-distribution homology detection AUPRC for remote homologs. In all cases, these metrics are computed on the validation splits, which have no overlap with the held-out test set.

S2 Supplementary Results

S2.1 Ablation study

We implemented six variations of DEDAL, each aiming to probe the effect on alignment and homology detection performance of one specific aspect of the model. These share training protocol and hyperparameters with the original DEDAL model, when applicable. To further reduce the computational footprint of these experiments, all models, including the original, were trained for half the number of steps and a single replicate was used for each ablation. Figures S4 and S5 display the alignment F_1 scores and homology detection AUPRC values for all approaches under study.

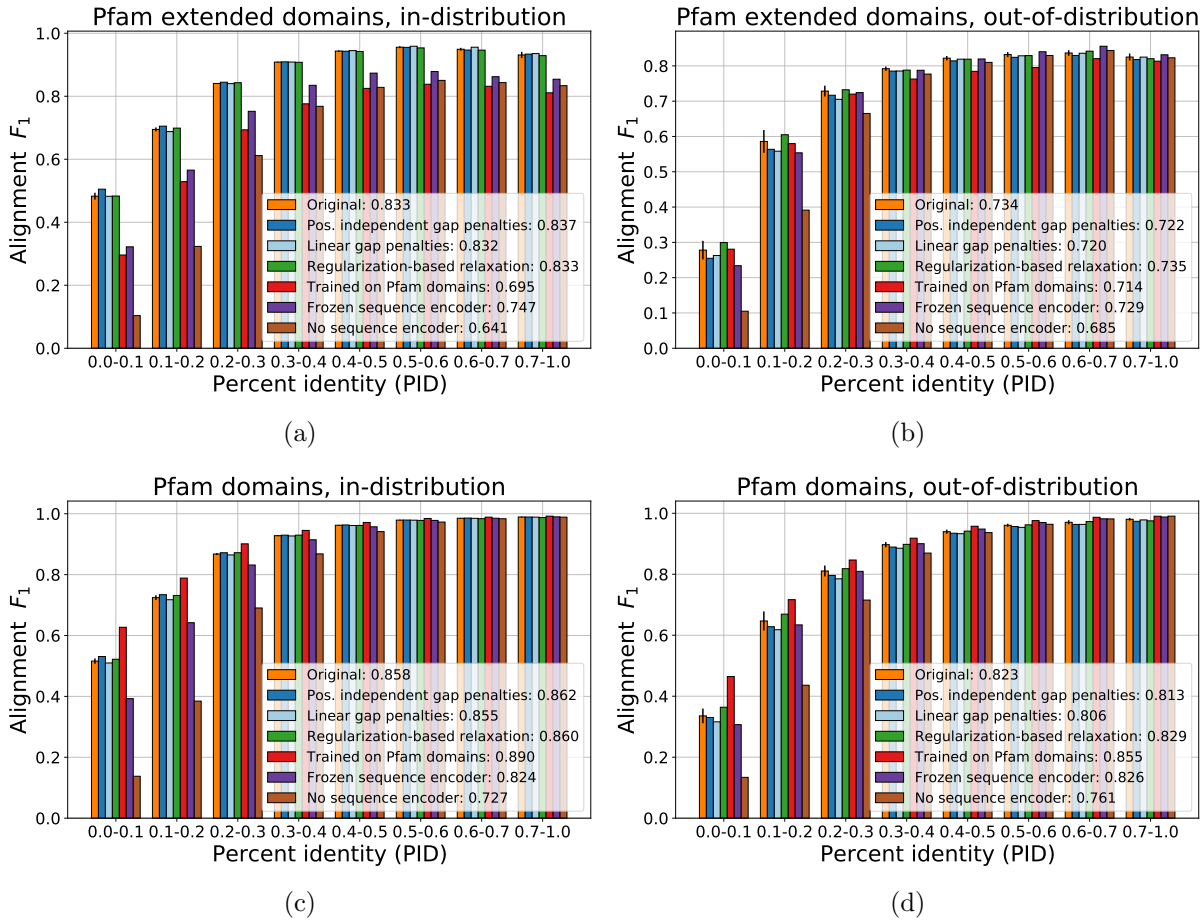


Figure S4: Alignment F_1 score of the original DEDAL model alongside six ablations. Overall performance (not stratified by PID) is displayed in the legend. Results for the original model were averaged over 10 replicates, with error bars describing 95% confidence intervals. Ablation results are based on a single replicate for computational considerations. (a) Pfam extended domains, in-distribution split. (b) Pfam extended domains, out-of-distribution split (held-out Pfam clans). (c) Pfam domain, in-distribution split. (d) Pfam domain, out-of-distribution split (held-out Pfam clans).

Position-independent gap penalties. Unlike DEDAL, traditional parameterizations of the SW algorithm use unique gap open and gap extend penalties that are shared across all positions. We tested a version of DEDAL that learns scalar, position-independent gap open and gap extend parameters instead of computing these as a function of the residue embeddings. Our findings suggest that this variant performs within the margin of error of the original model, achieving

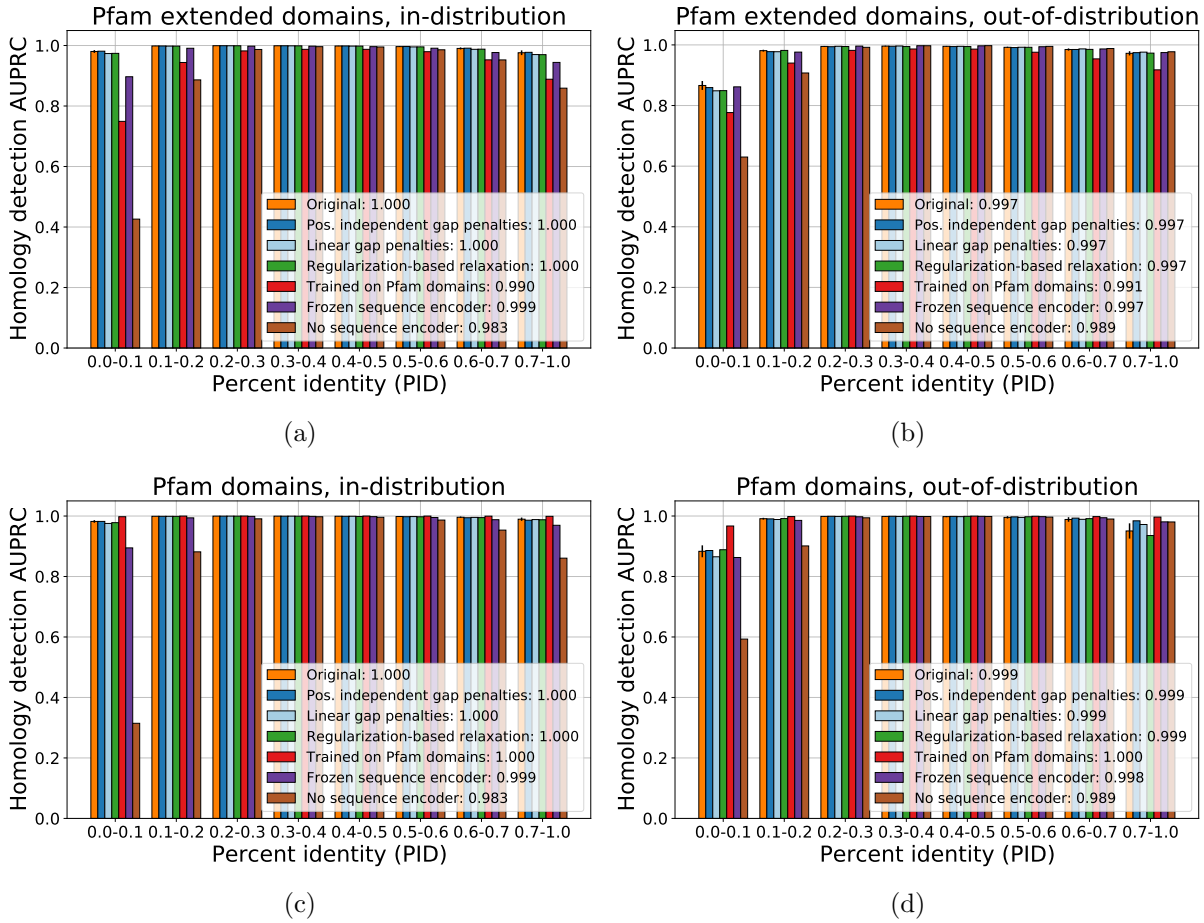


Figure S5: Homology detection AUPRC of the original DEDAL model alongside six ablations. Overall performance (not stratified by PID) is displayed in the legend. Results for the original model were averaged over 10 replicates, with error bars describing 95% confidence intervals. Ablation results are based on a single replicate for computational considerations. (a) Pfam extended domains, in-distribution split. (b) Pfam extended domains, out-of-distribution split (held-out Pfam clans). (c) Global alignment, in-distribution split. (d) Global alignment, out-of-distribution split (held-out Pfam clans).

slightly superior alignment F_1 scores in the in-distribution split yet marginally inferior results for out-of-distribution sequence pairs.

Linear gap penalty model. DEDAL uses an affine gap penalty model, allowing it to assign different costs to opening or extending gaps in an alignment. We explored an alternative version of DEDAL based on a linear gap penalty model instead. This simplification leads to a small performance drop for remote homologs that is most pronounced in the out-of-distribution split.

Regularization-based relaxation. In this work, we proposed two different techniques to relax the SW algorithm during training for end-to-end differentiability. While the final DEDAL model makes use of perturbations [7] to accomplish this, we also experimented with a variant based on smoothing via regularization [11]. Our results hint that both approaches perform comparably well, with the regularization-based version of DEDAL having slightly better alignment F_1 scores and somewhat inferior homology detection AUPRC values, albeit not by a statistically significant margin.

Trained on Pfam domains. We tried training DEDAL on the original Pfam-A seed sequences, whose ground-truth alignments are predominantly global or close to global, instead of on the extended Pfam domains we use to train DEDAL normally. As expected, this training regime enhances the performance of DEDAL when aligning Pfam domains even further, with in-distribution F_1 scores for the smallest PID bin (< 0.1) being up to 311% superior to those of the best-performing substitution matrix baseline. However, this comes at the cost of poor generalization in scenarios where the model must predict local alignments, such as when aligning extended Pfam domains, where it only outperforms substitution matrices by a moderate margin (81% as opposed to 219% for PID below 0.1). In contrast, when DEDAL is trained on the more diverse set of extended Pfam domains, it performs consistently well both in situations requiring local alignments and those for which the sought-after alignments happen to be approximately global.

Frozen sequence encoder. To train DEDAL, we jointly tune the parameters of the transformer encoder network that continuously embeds input sequences and the (differentiable) alignment layer that scores and aligns them. To examine the importance of end-to-end, joint training, we first fit a transformer encoder network on a masked language modelling task and then tuned the differentiable alignment layer's parameters while keeping the sequence encoder's weights constant. We find this to have a small, mostly negative effect on out-of-distribution performance. However, in-distribution performance is significantly affected, with e.g. alignment F_1 scores dropping by up to 33% for the hardest setting (PID < 0.1).

No sequence encoder. Finally, we take a step further relative to the previous ablation and eliminate the sequence encoder altogether. This setting is equivalent to using our differentiable alignment layer and training scheme to learn a substitution matrix alongside scalar gap open and gap extend penalties. Unsurprisingly, we observed this to perform comparably, if not somewhat worse, than the best-performing substitution matrices from the literature.

All in all, our ablation study indicates that DEDAL is surprisingly robust to changes in many aspects of the model. Notably, the specific way in which gap penalties are parameterized appears to have only a small effect on performance, as does the choice of approach to relax the SW algorithm during training. In contrast, we found end-to-end, joint training of a flexible sequence encoder and the differentiable alignment layer to be instrumental in realizing DEDAL's full potential.

S2.2 Results on the TAPE benchmark

We follow the Tasks Assessing Protein Embeddings (TAPE) benchmark [12] to probe whether using pairwise sequence alignment as supervision leads to better sequence embeddings for downstream tasks. TAPE provides standardized train, validation and test splits for five heterogeneous tasks. In a nutshell, these tasks are:

Secondary structure: A per-residue, 3-way (helix, strand, other) classification task, using the data from [13, 14]. It consists of $8,678$, $2,170$ and 513 sequences for training, validation and testing, respectively.

Residue-residue contacts: This task requires predicting residue pairs that are "in contact", defined as being less than 8Å apart. It relies on the data from [15], which provides $25,299$ training sequences alongside a test set of 40 sequences (CASP12 [16]) and an additional set of 224 sequences for validation.

Fold classification: A per-sequence, $1,195$-way classification task aiming to predict the SCOP hierarchy fold [17] each protein belongs to. Data is taken from [18] and is divided into $12,312$ training, 736 validation and 718 test sequences.

Fluorescence landscape: A per-sequence regression task whose goal is to predict the log-fluorescence intensity of variations of a green fluorescent protein (GFP). It contains $21,446$ training, $5,362$ validation and $27,217$ testing sequences obtained from [19].

Stability landscape: Another per-sequence regression task aiming to predict a proxy for the intrinsic stability of proteins. It uses $53,614$ training, $2,512$ validation and $12,851$ testing sequences from [20].

We aim to reproduce the experimental setup used in the TAPE benchmark suite [12]. Namely, we use the same output head and losses as [12] for most tasks and fine-tune the sequence encoder while training the output head. Rather than exactly following TAPE, we chose to simplify the output heads for the secondary structure and residue-residue contact prediction tasks for computational considerations. Concretely, for secondary structure prediction, we did not employ NetSurfP-2.0 [13]. Instead, we used a simpler model consisting of a stack of two 1D convolutional layers followed by an affine mapping with output dimension three, corresponding to the number of classes (helix, strand, other). The convolutional layers have 256 and 128 filters respectively, with kernel size 10, ReLU activations and dropout with rate 0.1. Similarly, for contact prediction we simplify the RaptorX-based output head [21] used in TAPE by (i) applying linear dimensionality reduction layer that halves the dimensionality of the residue embeddings and (ii) reducing the number of ResNet blocks from 30 to 6.

For each task in the TAPE benchmark suite, we consider three different initializations for the sequence encoder: (i) random, (ii) pretrained on masked language modelling only and (iii) pretrained with DEDAL. In all cases, we use the Adam optimizer with a fixed learning rate, treated as a hyperparameter to choose from $\{10^{-3}, 10^{-4}, 10^{-5}\}$, and apply early stopping to select the number of training steps. Both are tuned independently for each task to maximize the task's corresponding metric on its validation set.

Task	**Measure**	**Random**	**Language modelling**	**DEDAL**
SS	Accuracy	0.700 (0.001)	0.755 (0.001)	**0.756** (0.001)
Contact	AUPRC	0.158 (0.008)	**0.380** (0.020)	0.375 (0.024)
Folds	Accuracy	0.096 (0.009)	0.232 (0.011)	**0.240** (0.008)
Fluorescence	MSE	0.413 (0.132)	**0.153** (0.009)	0.198 (0.040)
Stability	MSE	**0.213** (0.027)	0.296 (0.052)	0.294 (0.056)

Table S2: Performance on downstream tasks from the TAPE benchmark suite for different initializations of the sequence encoder: (i) random (from scratch), (ii) pretrained on masked language modelling only and (iii) pretrained on alignment, homology detection and masked language modelling (DEDAL). The secondary structure (SS) and fold classification (Folds) tasks are evaluated in terms of classification accuracy (higher is better). The protein engineering tasks (fluorescence and stability) are quantified according to the mean squared error (MSE, lower is better). Finally, the residue-residue contact prediction task (Contact) uses AUPRC for medium range contacts (higher is better). All results are averaged over 10 replicates, with the standard deviation indicated in parenthesis.

The results for all five problems are summarized in Table S2. We find that both the masked language modelling and DEDAL succeed in learning sequence representations that are transferable to downstream tasks. Indeed, initializing the model from scratch is only the superior strategy in one of the five problems, stability landscape prediction, which happens to be that which has the largest amount of training data. When it comes to comparing masked language modelling to DEDAL, our results suggest that both strategies perform similarly, with the DEDAL-based initialization being marginally better at secondary structure prediction and fold classification, and marginally worse at predicting contacts and the fluorescence landscape

S3 Supplementary Figures

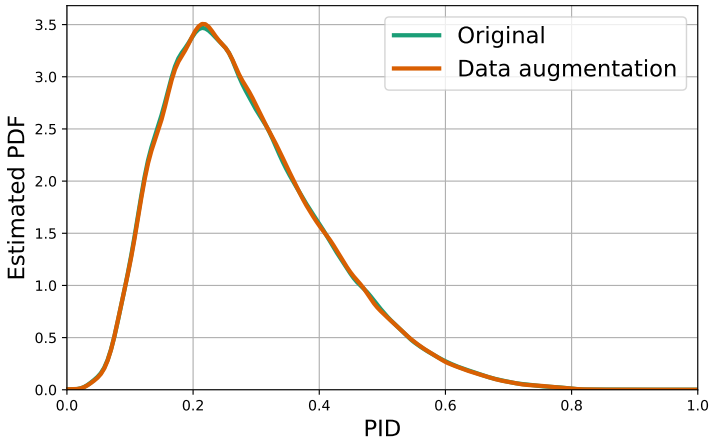


Figure S6: A kernel density estimate of the PID distribution for sequence pairs with and without data augmentation. 128,000 sequence pairs from the training set were used to estimate each distribution.

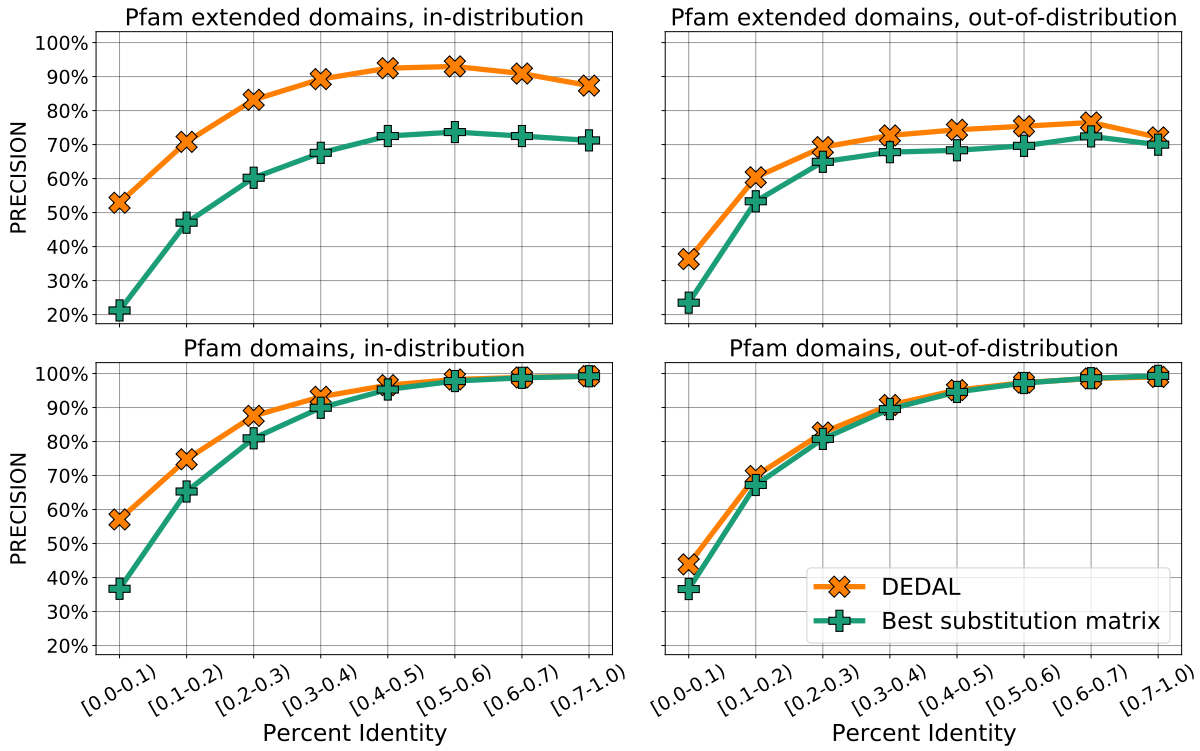


Figure S7: Alignment precision of DEDAL and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows).

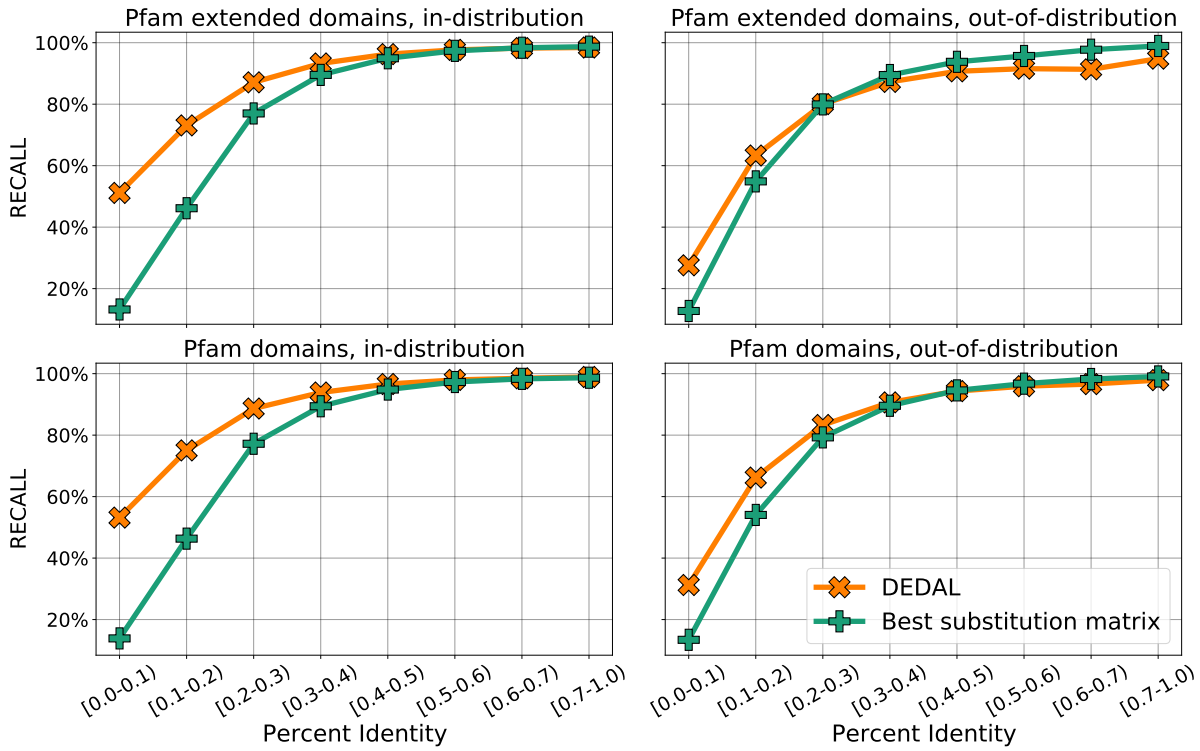


Figure S8: Alignment recall of DEDAL and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows).

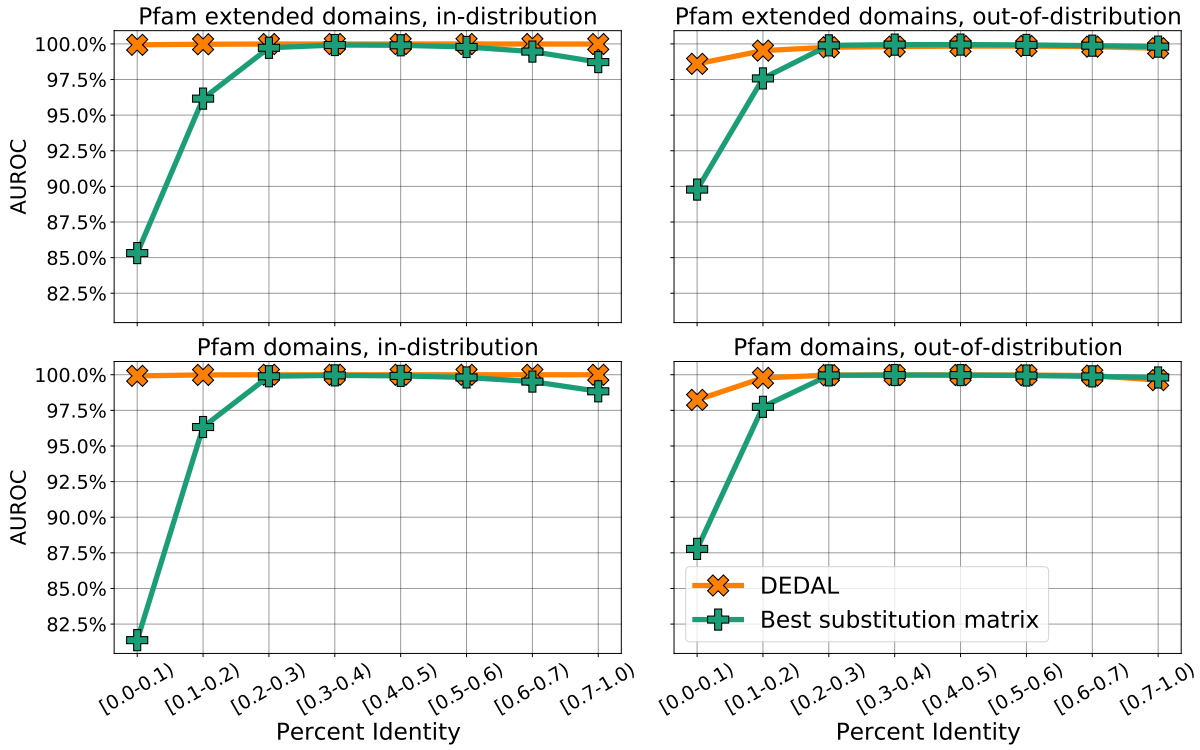


Figure S9: Homology detection AUROC of DEDAL and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows).

References

1. Vaswani, A. *et al. Attention is all you need* in *Advances in neural information processing systems 30* (eds Guyon, I. *et al.*) (Curran Associates, Inc., 2017), 5998–6008.
2. Rives, A. *et al.* Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proc. Natl. Acad. Sci. U.S.A.* **118,** e2016239118 (2021).
3. Ba, J. L., Kiros, J. R. & Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).
4. Xiong, R. *et al. On layer normalization in the transformer architecture* in *International Conference on Machine Learning* (2020), 10524–10533.
5. Hendrycks, D. & Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).
6. Glorot, X. & Bengio, Y. *Understanding the difficulty of training deep feedforward neural networks* in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), 249–256.
7. Berthet, Q. *et al. Learning with differentiable perturbed optimizers* in *Advances in neural information processing systems 33* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H.) (2020).
8. Abadi, M. *et al.* Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
9. Bradbury, J. *et al. JAX: composable transformations of Python+NumPy programs* version 0.2.5. 2018. `http://github.com/google/jax`.

10. Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* **32,** 8026–8037 (2019).

11. Mensch, A. & Blondel, M. *Differentiable dynamic programming for structured prediction and attention* in *Proceedings of the 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) **80** (PMLR, 2018), 3462–3471.

12. Rao, R. *et al.* Evaluating protein transfer learning with TAPE. *Advances in neural information processing systems* **32,** 9689 (2019).

13. Klausen, M. S. *et al.* NetSurfP-2.0: Improved prediction of protein structural features by integrated deep learning. *Proteins: Structure, Function, and Bioinformatics* **87,** 520–527 (2019).

14. Cuff, J. A. & Barton, G. J. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins: Structure, Function, and Bioinformatics* **34,** 508–519 (1999).

15. AlQuraishi, M. ProteinNet: a standardized data set for machine learning of protein structure. *BMC Bioinform.* **20,** 1–10 (2019).

16. Moult, J., Fidelis, K., Kryshtafovych, A., Schwede, T. & Tramontano, A. Critical assessment of methods of protein structure prediction (CASP)—Round XII. *Proteins: Structure, Function, and Bioinformatics* **86,** 7–15 (2018).

17. Chandonia, J.-M., Fox, N. K. & Brenner, S. E. SCOPe: classification of large macromolecular structures in the structural classification of proteins—extended database. *Nucleic Acids Res.* **47,** D475–D481 (2019).

18. Hou, J., Adhikari, B. & Cheng, J. DeepSF: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics* **34,** 1295–1303 (2018).

19. Sarkisyan, K. S. *et al.* Local fitness landscape of the green fluorescent protein. *Nature* **533,** 397–401 (2016).

20. Rocklin, G. J. *et al.* Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science* **357,** 168–175 (2017).

21. Peng, J. & Xu, J. RaptorX: exploiting structure information for protein alignment by statistical inference. *Proteins: Structure, Function, and Bioinformatics* **79,** 161–171 (2011).