
Deep embedding and alignment of protein sequences

In the format provided by the
authors and unedited

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 (c.f. “Trained on Pfam domains” ablation in Figures S5 and S6).

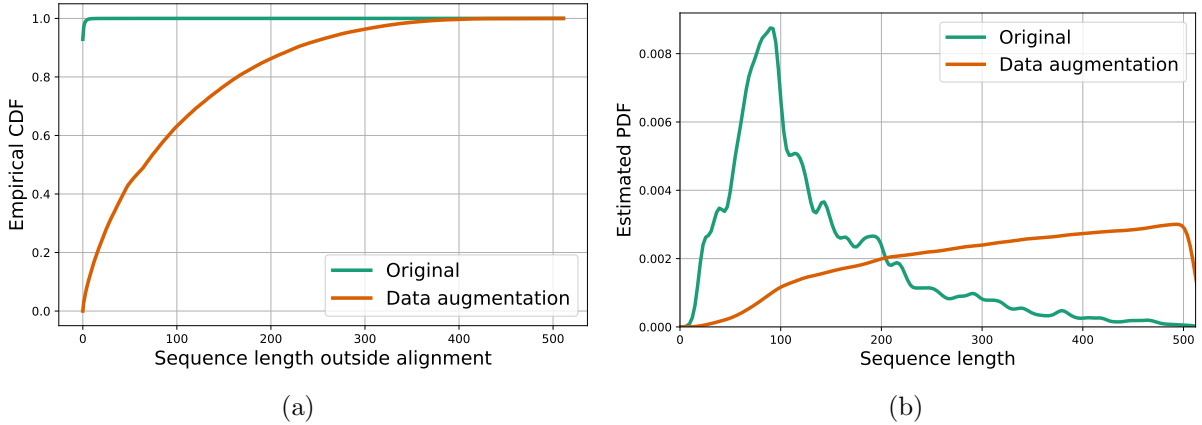


Figure S1: (a) The empirical CDF of the number of residues not mapping to matches or internal gaps in a target alignment. (b) A kernel density estimate of the sequence length distribution. A random set of 1,048,576 sequence pairs from the training set was used for both figures.

To overcome this limitation, we apply *data augmentation* to the Pfam-A seed dataset, creating *local* pairwise sequence alignment targets by adding flanking sequences to the original Pfam domains. To this end, every time a Pfam-A seed sequence is drawn as part of a sequence pair for training or evaluation, we first sample lengths for the N-terminus and C-terminus flanks to be added as

$$\begin{aligned} L_{\text{flanks}} &\sim U(0, L_{\text{model}} - L_{\text{seq}}), \\ L_{\text{N}} &\sim U(0, L_{\text{flanks}}), \\ L_{\text{P}} &= L_{\text{flanks}} - L_{\text{N}}, \end{aligned} \tag{1}$$

with L_{seq} being the length of the original Pfam-A seed domain sequence and $L_{\text{model}} = 511$ the maximum sequence length we consider for this task. In other words, the total length of the flanking sequences being added is sampled uniformly between zero and the maximum possible, given the computational budget. These L_{flanks} residues are subsequently allocated to N and C-terminus uniformly at random among all valid choices. Next, for each of the two flanks, we sample a random (sub)sequence of length L_{N} (resp. L_{P}) from UniProtKB. To prevent the inclusion of flanks from causing any test set “leakage”, we ensure that:

1. UniProtKB sequences that match any of the Pfam families in the out-of-distribution validation or out-of-distribution test splits may *not* be used to generate flanks.
2. The same UniProtKB sequence may *not* be used to generate flanks for Pfam-A seed sequences in different splits.

- UniProtKB sequences that contain Pfam-A seed domain sequences may only be used to generate flanks for Pfam-A seed sequences in the same split to which those sequences were allocated. As a corollary, UniProtKB sequences containing two or more Pfam-A seed domain sequences that form part of different splits may *not* be used to generate flanks..

Furthermore, we aim to minimize the number of erroneous “ground-truth” extended alignments by guaranteeing that, for every pair of extended Pfam domain sequences, the flanks added to one sequence have no known clan annotations in common with neither the Pfam-A seed region nor the flanks of the sequence. In practice, we rely on the entries of the `pfamA_reg_full_significant`, `uniprot_reg_full` and `pfamA_reg_seed` tables in Pfam’s SQL database as the source of clan annotations for UniProtKB sequences. To further increase the data diversity, the N and C-terminus flank lengths as well as the flanking sequences themselves are sampled independently for each sequence pair. In other words, the same Pfam-A seed domain sequence may be extended differently depending on the other sequence it is paired with.

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.

The correctness of the proposed data augmentation scheme is partly limited by the coverage of existing clan-level annotations. Because of this, it is possible that a pair of Pfam-A seed domain sequences is extended with flanks that are related by a not-yet-known homology relationship. To corroborate that such occurrences are not prevalent enough to affect our results in any significant manner, we also experimented with an alternative data augmentation scheme using *synthetic* flanks. These are simply obtained by sampling residues i.i.d. using the background amino acid frequencies estimated from Pfam-A seed sequences. The resulting Pfam synthetic domains are arguably less realistic, as well as challenging for DEDAL, which was trained specifically to align “protein-like” sequences. In spite of this, Figures S2 and S3 show that the performance of DEDAL degrades only slightly when evaluating on sequences with synthetic flanks, despite having been trained only on “protein-like” sequences with UniProt-based flanks. Moreover, the baselines perform similarly with both types of flanking sequences, suggesting that erroneous ground-truth alignments due to missing annotations, while possible, are rare enough not to compromise our evaluation.

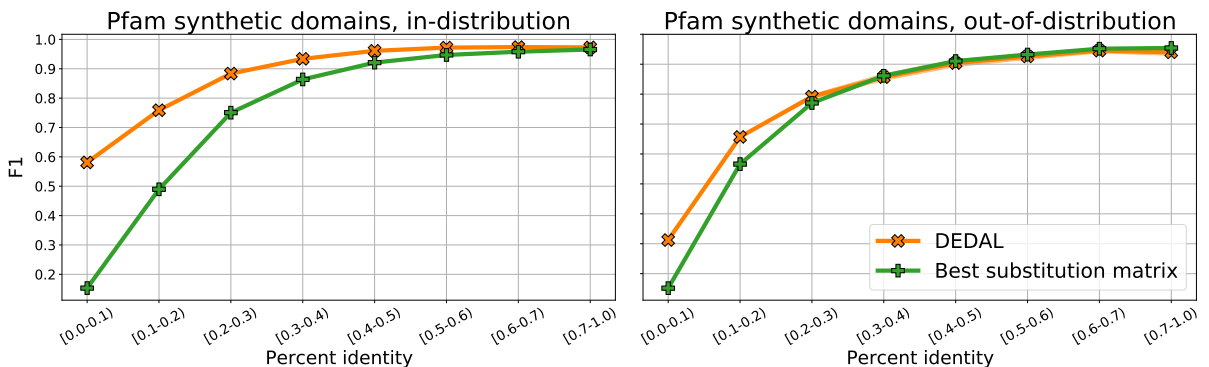


Figure S2: Alignment F_1 score of DEDAL (trained on UniProt-based flanks) and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns) for Pfam domains extended with synthetically-generated flanks (sampled i.i.d. from the background amino acid-frequencies in Pfam-A seed). Results for DEDAL were averaged over 10 replicates, with error bars describing 95% confidence intervals.

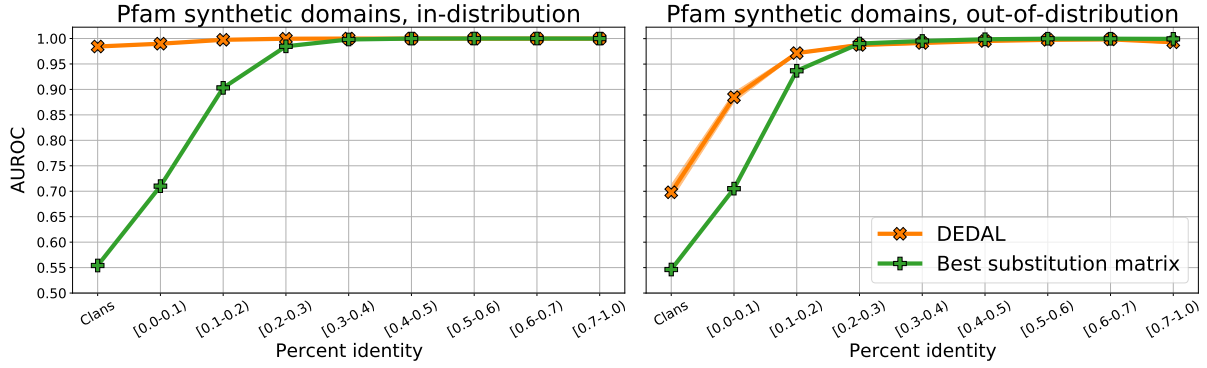


Figure S3: Homology detection AUROC of DEDAL (trained on UniProt-based flanks) and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns) for Pfam domains extended with synthetically-generated flanks (sampled i.i.d. from the background amino acid-frequencies in Pfam-A seed). Results for DEDAL were averaged over 10 replicates, with error bars describing 95% confidence intervals.

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 $x = (x_1, x_2, \dots, x_n)$ over a finite alphabet ($x_i \in \mathcal{A}$ for all $i = 1, \dots, n$) to a sequence of d -dimensional vectors of the same length $T_\phi(x) = (T_{\phi,1}(x), T_{\phi,2}(x), \dots, T_{\phi,n}(x))$. 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 $\mathcal{A} = \llbracket 1, k \rrbracket$ 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 $\langle \text{EOS} \rangle$, which we append to each sequence, and $\langle \text{MASK} \rangle$, 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 $x = (x_1, x_2, \dots, x_n) \in \mathcal{A}^n$, 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

$$Z_0 = XF + P, \quad (2)$$

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],

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

where $\circ$ denotes function composition.

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

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

where $W_i^Q, W_i^K, W_i^V : \mathbb{R}^d \rightarrow \mathbb{R}^{d/h}$ and $W_i^O : \mathbb{R}^{d/h} \rightarrow \mathbb{R}^d$ 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,

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

where $W^{\text{in}} : \mathbb{R}^d \rightarrow \mathbb{R}^{4d}$ and $W^{\text{out}} : \mathbb{R}^{4d} \rightarrow \mathbb{R}^d$ 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 $x \in \mathcal{A}^n$,

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

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,

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

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. In all experiments, a global batch size of 128 sequences (resp. sequence pairs) was used for the masked language modelling and alignment tasks while the homology task used 256 sequence pairs. 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 $\lambda_{\text{HD}} = \lambda_{\text{LM}} = 20$. 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 $i = 1, \dots, n$ 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:

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

where $\max_\tau(\{u_1, \dots, u_k\}) := \tau \log \sum_{i=1}^k e^{u_i/\tau}$ 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 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.

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. $-\infty$).

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 $k \in \llbracket 1, n + m - 1 \rrbracket$. To reduce the number of non-contiguous indexing operations, we define mappings $f^{(+)}, f^{(-)} : \mathbb{R}^{b \times n \times m} \rightarrow \mathbb{R}^{b \times n \times n + m - 1}$ to rearrange the inputs S, D and E mapping their anti-diagonals to a single axis in the output tensors. Namely,

$$f_{s,i,k}^{(+)}(A) = \begin{cases} A_{s,i,k-i-1}, & \text{if } k - i - 1 \in \llbracket 1, m \rrbracket, \\ +\infty, & \text{otherwise,} \end{cases} \quad (9)$$

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 $\tilde{S} := f^{(-)}(S)$, $\tilde{D} := f^{(+)}(D)$ and $\tilde{E} := f^{(+)}(E)$, we can rewrite the dynamic program updates in Equation 8 as

$$\begin{aligned} M_{i,k} &= \max_{\tau} (0, M_{i-1,k-2}, X_{i-1,k-2}, Y_{i-1,k-2}) + \tilde{S}_{i,k}, \\ X_{i,k} &= \max_{\tau} (M_{i-1,k-1} - \tilde{D}_{i,k}, X_{i-1,k-1} - \tilde{E}_{i,k}, Y_{i-1,k-1} - D_{i,j}), \\ Y_{i,k} &= \max_{\tau} (M_{i,k-1} - \tilde{D}_{i,k}, Y_{i,k-1} - \tilde{E}_{i,k}), \end{aligned} \quad (10)$$

which can be trivially vectorized along the batch and i axes, requiring only a single `for` loop on $k \in \llbracket 1, n + m - 1 \rrbracket$. 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 S4.

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 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 $T := \max_{i,j} \left(0, \max_{\tau} (M_{i,j}) \right)$ such that $s_{\tau}^*(x, y) = T$. Our derivation will proceed in

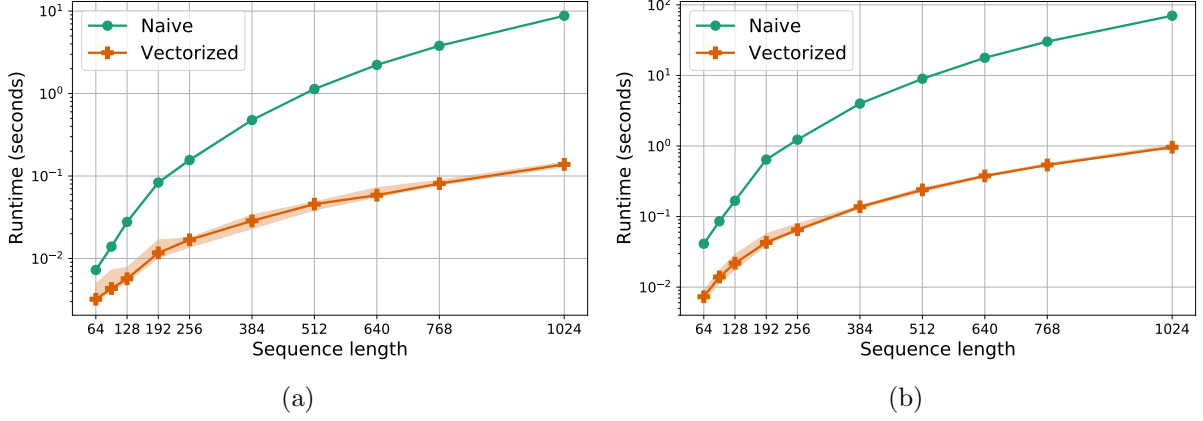


Figure S4: Runtime to align a batch of sequence pairs as a function of the length of the sequences for the proposed vectorization approach and a naive baseline implemented as a double `for` loop. Times shown reflect the forward pass only, computed in inference mode ($\tau = 0$) using 4 TPU v3 chips. 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.

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

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

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

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

where we used that

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

and that

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

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

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

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 $\max_\tau(\{u_1, \dots, u_k\}) = \tau \log \sum_{i=1}^k e^{u_i/\tau}$ whose partial derivatives $\frac{\partial \max_\tau(\{u_1, \dots, u_k\})}{\partial u_i} = \exp\left(\frac{u_i - \max_\tau(\{u_1, \dots, u_k\})}{\tau}\right)$ 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 metrics¹ 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

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

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 $k^* = \operatorname{argmax}_k f_k$. 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.

¹For 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.

S2 Supplementary Results

S2.1 Ablation study

We implemented seven 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. Unlike for all other ablations and the original DEDAL model, we found that disabling the homology loss for the first 300,000 steps was detrimental for the performance of the ablation trained on the original Pfam domains. Thus, we omit this solely for this ablation. To further reduce the computational footprint of these experiments a single replicate was used for each ablation. Figures S5 and S6 display the alignment F_1 scores and homology detection AUROC values for all approaches under study, which we discuss below.

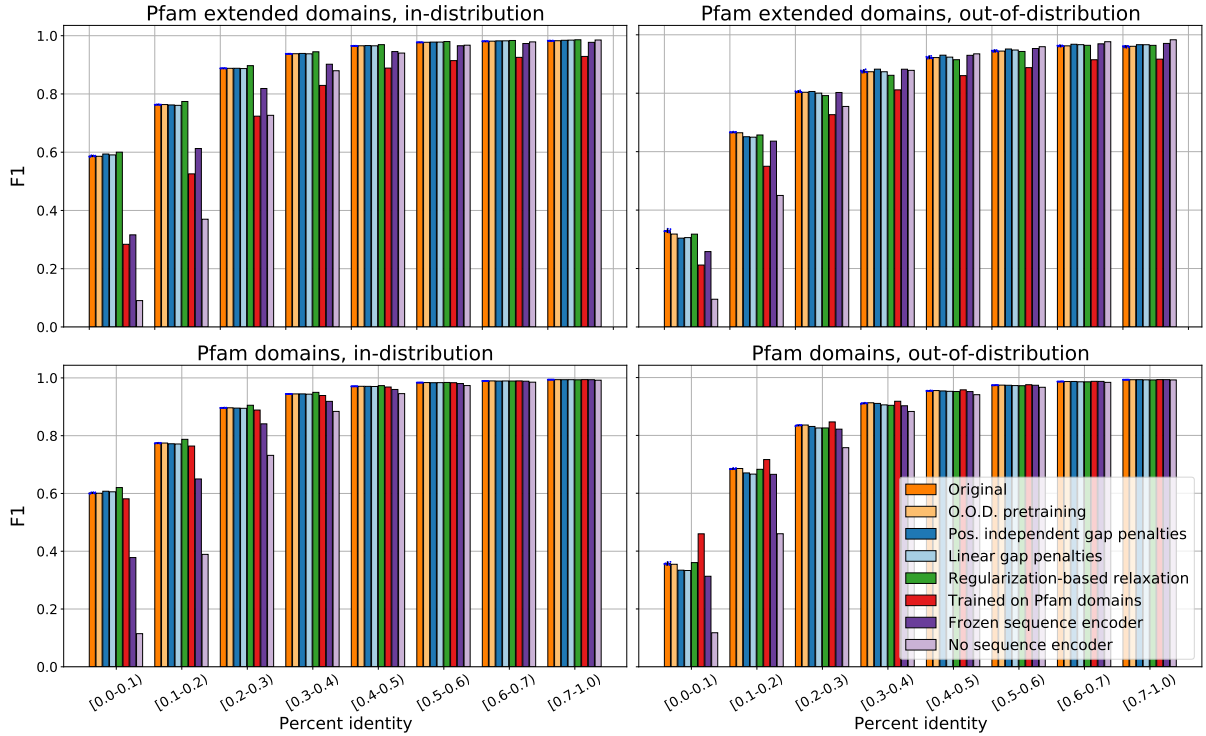


Figure S5: Alignment F_1 score of the original DEDAL model alongside seven ablations in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows). Results for the original model were averaged over 10 replicates, with error bars describing 95% confidence intervals. Ablation results are based on a single replicate for computational considerations.

O.O.D. pretraining. The original DEDAL model is trained on a subset of UniRef50 that contains sequences matching any Pfam families, including those in the out-of-distribution splits. Therefore, out-of-distribution performance in this regime corresponds to the case where a user wants to align sequences without any known ground-truth alignments or homologs, but which are nevertheless part of the known “protein universe”, as represented by increasingly large databases (such as UniRef50). We believe this to be the most relevant use-case in practice. However, to study the extent to which the performance of DEDAL in the out-of-distribution regime would degrade if these sequences were also excluded from pretraining, we re-trained the model using a new split of UniRef50 that simulates this more challenging scenario. To this end, we used the `hmmsearch` utility from HMMER v3.3.2 [11] to search for matches of any of the

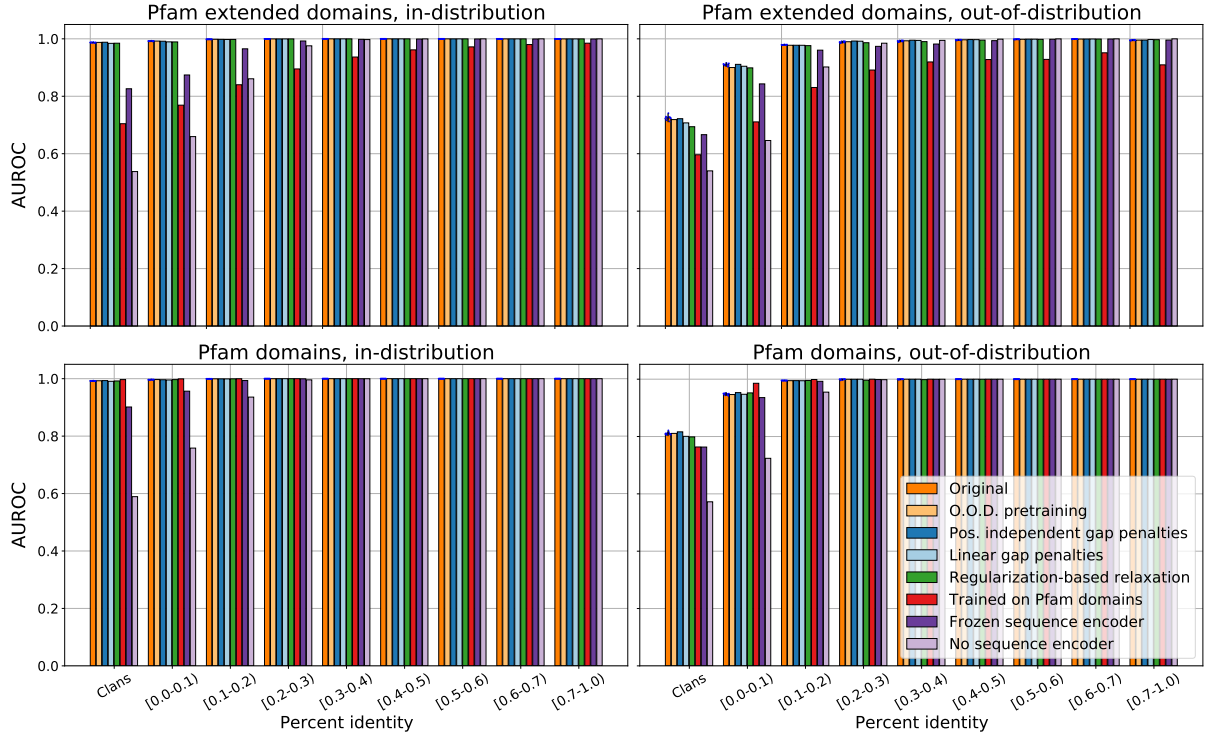


Figure S6: Homology detection AUROC of the original DEDAL model alongside seven ablations in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows). Results for the original model were averaged over 10 replicates, with error bars describing 95% confidence intervals. Ablation results are based on a single replicate for computational considerations.

955 Pfam clans in the out-of-distribution test set against the UniRef50 sequences at an E-value threshold of 0.01. This resulted in 1,279,217 UniRef50 sequences with significant matches to at least one Pfam family in the out-of-distribution test set, all of which were allocated to the test set for masked language modeling pretraining of this new split of UniRef50. Among the remaining 28,882,894 sequences with no significant matches to Pfam families in the out-of-distribution test set, we held-out an additional 90,000 and 1,736,994 sequences for validation and test, respectively. These were chosen uniformly at random with the goal of making the new splits comparable in size to those used to train the original DEDAL model. All in all, we found this ablation to behave similarly to the original, exhibiting only a small performance degradation for remote homologs ($PID < 0.1$), with the alignment F_1 score and homology detection AUROC dropping from 0.329 to 0.318 and from 0.910 to 0.900, respectively.

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 slightly inferior alignment F_1 scores for remote homologs in the out-of-distribution split yet marginally superior results for sequences with $PID \geq 0.3$.

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 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 [12]. Our results hint that both approaches perform comparably well, with the regularization-based version of DEDAL performing slightly better in-distribution and slightly worse out-of-distribution.

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 e.g. out-of-distribution F_1 scores for the smallest PID bin (< 0.1) being up to 143% 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 (43% as opposed to 119% 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 mostly negative effect on in-distribution performance that is most salient for remote homologs, with e.g. alignment F_1 scores dropping by up to 85% 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 [13] 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 [14, 15]. 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 [16], which provides 25,299 training sequences alongside a test set of 40 sequences (CASP12 [17]) 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 [18] each protein belongs to. Data is taken from [19] 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 [20].

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 [21].

We aim to reproduce the experimental setup used in the TAPE benchmark suite [13]. Namely, we use the same output head and losses as [13] 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 [14]. 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 [22] 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.

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, fold classification, and marginally worse at predicting contacts and the stability landscape.

Task	Measure	Random	Language modelling	DEDAL
SS	Accuracy	0.700 (0.001)	0.755 (0.001)	0.757 (0.001)
Contact	AUPRC	0.158 (0.008)	0.380 (0.020)	0.378 (0.009)
Folds	Accuracy	0.096 (0.009)	0.232 (0.011)	0.263 (0.011)
Fluorescence	MSE	0.413 (0.132)	0.153 (0.009)	0.150 (0.018)
Stability	MSE	0.213 (0.027)	0.296 (0.052)	0.309 (0.058)

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.

S3 Supplementary Figures

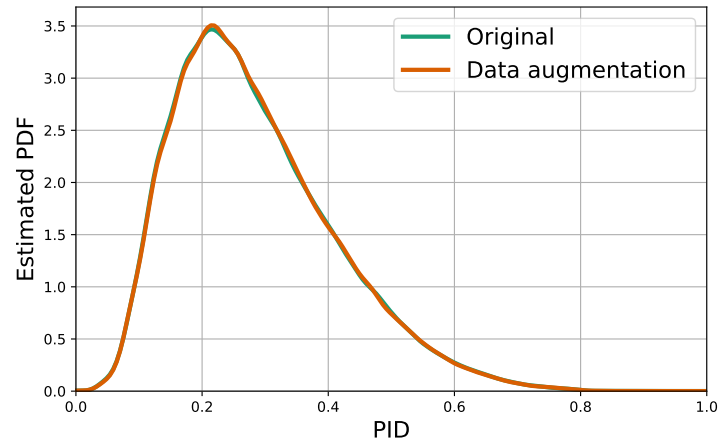


Figure S7: A kernel density estimate of the PID distribution for sequence pairs with and without data augmentation. 128,000 sequence pairs from the training set were used to estimate each distribution.

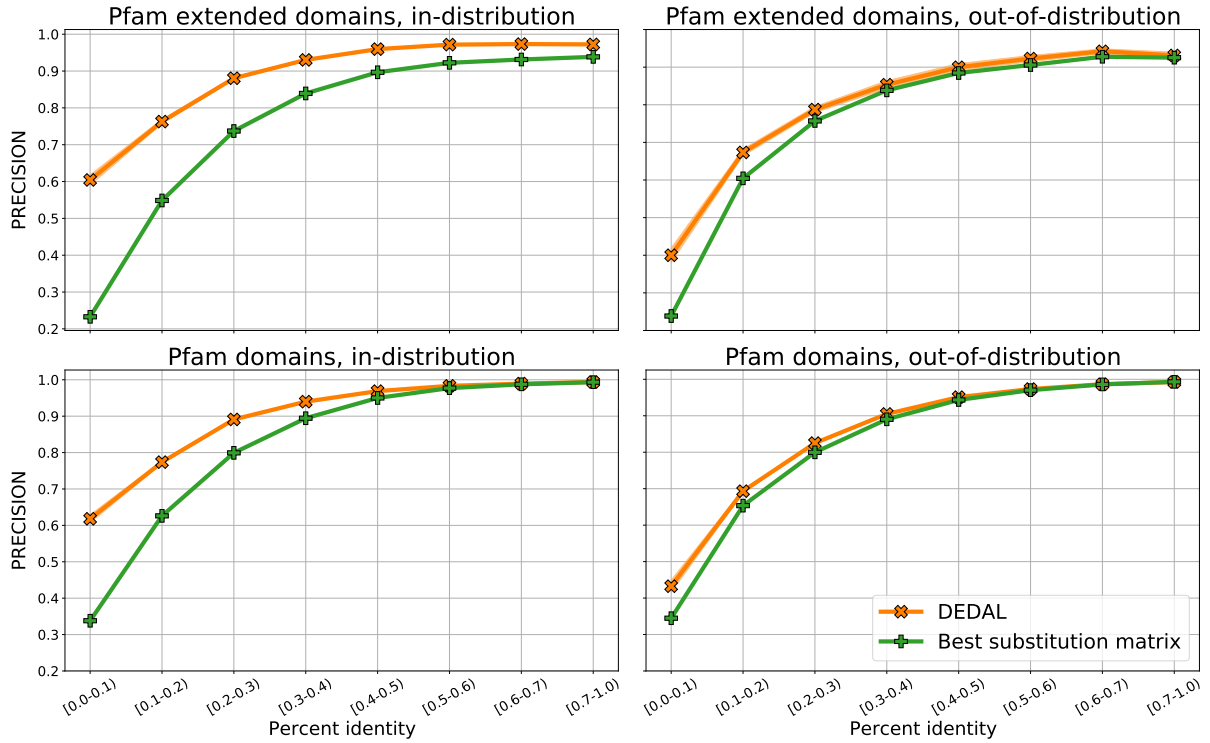


Figure S8: Alignment precision of DEDAL and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows). Results for DEDAL were averaged over 10 replicates, with error bars describing 95% confidence intervals.

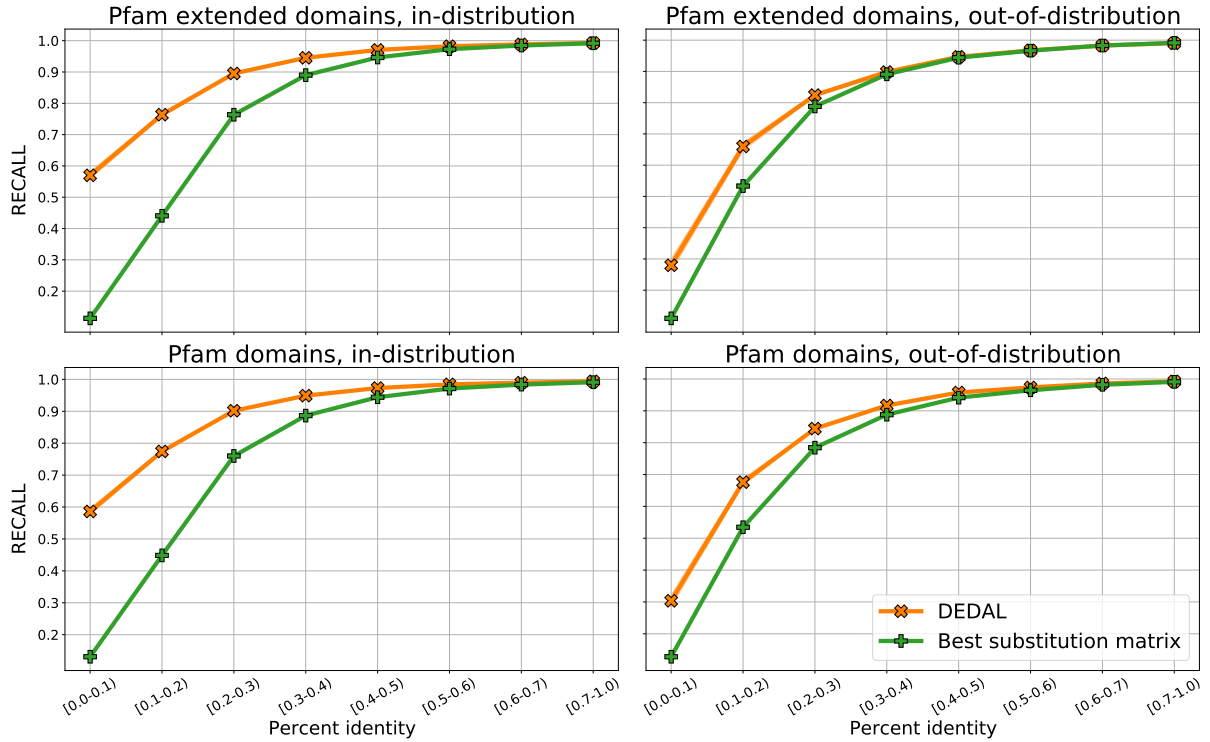


Figure S9: Alignment recall of DEDAL and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows). Results for DEDAL were averaged over 10 replicates, with error bars describing 95% confidence intervals.

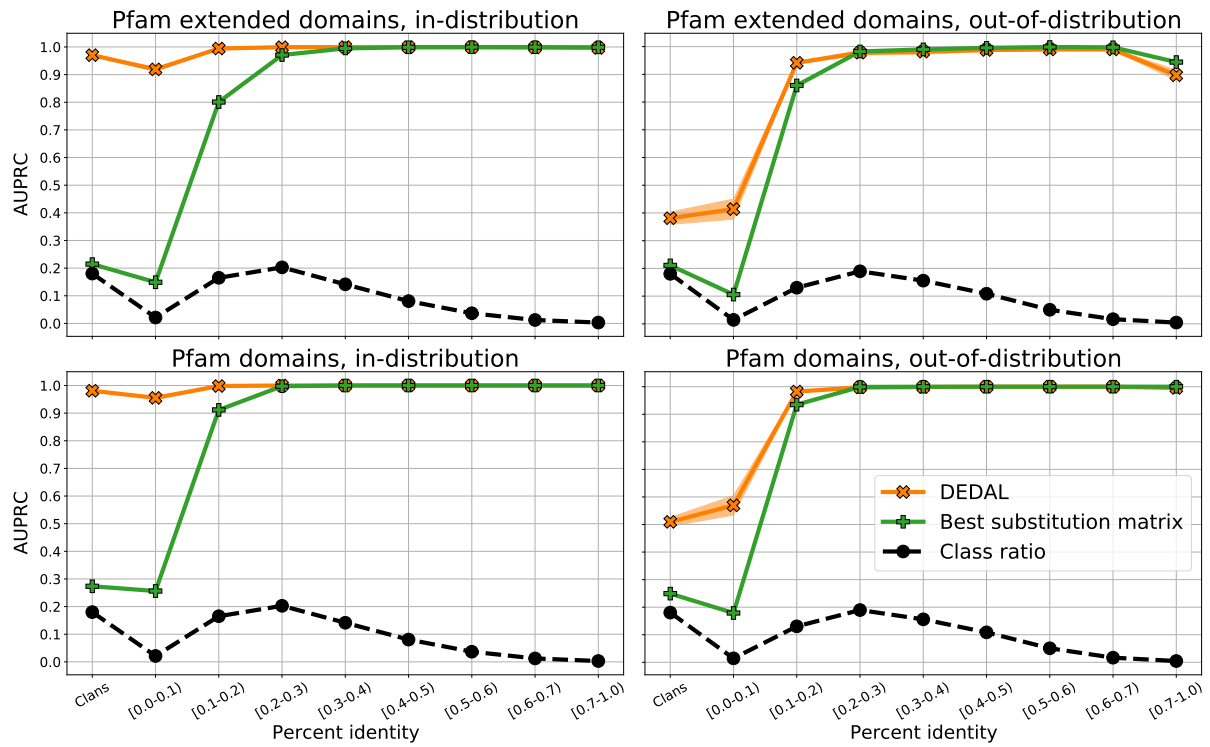


Figure S10: Homology detection AUPRC of DEDAL and the best-performing substitution matrix baseline, in the in- and out-of-distribution settings (respectively, left and right columns), and for Pfam extended or raw domains (respectively, top and bottom rows). Results for DEDAL were averaged over 10 replicates, with error bars describing 95% confidence intervals.

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. Eddy, S. R. Accelerated profile HMM searches. *PLoS computational biology* **7**, e1002195 (2011).
12. 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.
13. Rao, R. *et al.* Evaluating protein transfer learning with TAPE. *Advances in neural information processing systems* **32**, 9689 (2019).
14. 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).
15. 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).
16. AlQuraishi, M. ProteinNet: a standardized data set for machine learning of protein structure. *BMC Bioinform.* **20**, 1–10 (2019).
17. 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).
18. 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).

19. Hou, J., Adhikari, B. & Cheng, J. DeepSF: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics* **34**, 1295–1303 (2018).
20. Sarkisyan, K. S. *et al.* Local fitness landscape of the green fluorescent protein. *Nature* **533**, 397–401 (2016).
21. Rocklin, G. J. *et al.* Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science* **357**, 168–175 (2017).
22. Peng, J. & Xu, J. RaptorX: exploiting structure information for protein alignment by statistical inference. *Proteins: Structure, Function, and Bioinformatics* **79**, 161–171 (2011).