MULTIPLE SEQUENCE ALIGNMENT AS A SEQUENCE-TO-SEQUENCE LEARNING PROBLEM

Anonymous authors
Paper under double-blind review

ABSTRACT

The sequence alignment problem is one of the most fundamental problems in bioinformatics and a plethora of methods were devised to tackle it. Here we introduce BetaAlign, a novel methodology for aligning sequences using a natural language processing (NLP) approach. BetaAlign accounts for the possible variability of the evolutionary process among different datasets by using an ensemble of transformers, each trained on millions of samples generated from a different evolutionary model. Our approach leads to outstanding alignment accuracy, often outperforming commonly used methods, such as MAFFT, DIALIGN, ClustalW, T-Coffee, and MUSCLE.

1 INTRODUCTION

A multiple sequence alignment (MSA) provides a record of homology at the single position resolution within a set of homologous sequences. In order to infer the MSA from input sequences, one has to consider different evolutionary events, such as, substitutions and indels (i.e., insertions and deletions). MSAs can be computed for DNA, RNA, or amino acid sequences. MSA inference is considered one of the most common problems in biology (Van Noorden et al., 2014). Moreover, MSAs are required input for various widely-used bioinformatics methods such as domain analysis, phylogenetic reconstruction, motif finding, and ancestral sequence reconstruction (Kemena & Notredame, 2009). These methods assume the correctness of the MSA, and their performance might degrade when inaccurate MSAs are used as input (Ogden & Rosenberg, 2006; Privman et al., 2012).

MSA algorithms typically assume a fixed cost matrix as input, i.e., the penalty for aligning non-identical characters and the reward for aligning identical characters. They also assume a penalty for the introduction of gaps. These costs dictate the score of each possible alignment, and the algorithm aims to output the alignment with the highest score. MSA algorithms often allow users to select among several default scoring schemes and tune parameters that control the MSA computation. However, in practice, these parameters are seldom altered, and only a few default configurations are used. Previous studies demonstrated that fitting the cost matrix configuration to the data increases the MSA inference accuracy (Rubio-Largo et al., 2018; Llinares-López et al., 2021).

Alignment algorithms are often benchmarked against empirical alignment regions, which are thought to be reliable. However, such regions within alignments do not reflect the universe of alignment problems (Thompson et al., 1994). Of note, these regions were often manually computed, so there is no guarantee that they represent a reliable "gold standard" (Morrison, 2009). When testing alignment programs with simulated complex alignments, the results differ from the benchmarks results (Chang et al., 2014).

In the last decade, deep-learning algorithms have revolutionized various fields (LeCun et al., 2015), including computer vision (Voulodimos et al., 2018), natural language processing (NLP) (Young et al., 2018), sequence correction Baid et al. (2021), and medical diagnosis (Rakocz et al., 2021; Hill et al., 2021). Neural-network solutions often resulted in a substantial increase in prediction accuracy compared to traditional algorithms. Deep learning has also changed molecular biology and evolutionary research, e.g., by allowing accurate predictions of three-dimensional protein structures using AlphaFold (Jumper et al., 2021).

To the best of our knowledge, sequence-to-sequence learning is not currently used to align sequences. We propose BetaAlign, a novel deep-learning method that is trained on known align-

ments, and is able to accurately align novel sets of sequences. The method is based on a recent state-of-the-art deep-learning architecture designed for sequence-to-sequence tasks called "transformer" (Vaswani et al., 2017), which was originally developed for machine translation. BetaAlign was trained on millions of alignments drawn from different evolutionary models. We analyze the accuracy of BetaAlign and show that it is comparable, and in some cases superior to the most popular MSA algorithms, T-Coffee (Notredame et al., 2000), ClustalW (Larkin et al., 2007), DIALIGN (Morgenstern, 2004), MUSCLE (Edgar, 2004), MAFFT (Katoh & Standley, 2013) and PRANK (Löytynoja & Goldman, 2008).

BetaAlign converts the alignment problem into a translation problem, a well-studied problem within the NLP field (Hirschberg & Manning). To frame an alignment task as an NLP translation task, we represent both the unaligned sequences and the resulting alignment as sentences in two different "languages". We first present the NLP approach for pairwise alignment. We represent the two input sequences as a sentence in a language that we term "concat" (the source language). It is termed "concat" because, in this representation, each pair of input sequences is concatenated, with the pipe character ("|") representing the border between the sequences. In this language, each character in the resulting string is considered a different word. For example, when we aligned the sequence "AAG" against the sequence "ACGG", in the "concat" language, this will be represented by the sentence "A A G | A C G G" (Fig. 1). In this sentence, there are eight words. Of note, in the "concat" language there are only five possible unique words. A dictionary of a language is the entire set of words used in that language, and thus in the "concat" language, the dictionary is the set "A", "C", "G", "T", "|".

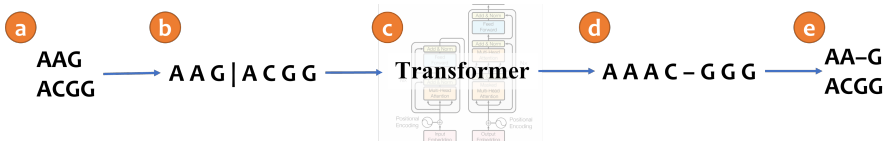

Figure 1: An illustration of aligning sequences with the sequence-to-sequence learning. (a) Consider two input sequences "AAG" and "ACGG". (b) The result of encoding the unaligned sequences into the source language ("concat" representation). (c) The sentence from the source language is translated to the target language via a transformer model. (d) The translated sentence in the target language ("spaces" representation). (e) The resulting alignment, decoded from the translated sentence, in which "AA-G" is aligned to "ACGG". The transformer architecture illustration is adapted from (Vaswani et al., 2017).

The target language, i.e., the language of the output alignment, is termed here "spaces". In this language, the dictionary is the set "A", "C", "G", "T", "–". In this representation, the words of the two different aligned sequences are interleaved. That is, the first two words in the "spaces" output sentence are the first character of the first and the second sequence, respectively. The third and fourth words correspond to the second column of the pairwise alignment, etc. In the above example, assume that in the output alignment "AA-G" is aligned to "ACGG", then, in the "spaces" language, this will be represented as the sentence "A A A C – G G G" (Fig. 1).

With these representations, the input sequences are a sentence in one language, the output alignment is a sentence in another language, and a perfect alignment algorithm is requested to accurately translate one language to the other. The details regarding the training and execution of the NLP transformers are provided in the Methods and Supplementary Information. Of note, the occasional generation of invalid alignments was encountered while developing BetaAlign. This challenge was addressed by introducing an ensemble of transformers (see Supplementary Information).

2 Materials and Methods

2.1 An NLP-Based Translation Model

Our hypothesis is that cutting-edge NLP tools can learn from known examples, and a trained model can infer accurate alignments for unseen data. A plethora of neural network architectures is available in the NLP domain for translation, including recurrent neural networks (Sutskever et al., 2014) and transformers (Vaswani et al., 2017). As these architectures are sequence to sequence, the input

to these models is a sentence, and the output is a predicted translation. Our approach relies on optimizing the model based on training data, i.e., the learning phase of the algorithm. In our case, a massive amount of training data, i.e., true alignments, is obtained using simulations. We simulate the training alignments using SpartaABC (Loewenthal et al., 2021). SpartaABC allows simulating datasets along a phylogenetic tree, with various indel-length distributions, and with different indel-to-substitution rate ratios. It allows finding indel model parameters that best describe a specific empirical dataset. It also assumes that the evolutionary dynamics of insertions and deletions are characterized by a different set of parameters, i.e., a rich-indel model (Loewenthal et al., 2021).

Formally, let x denote the input sequences and y denote the correct MSA. Each training data point is a pair (x, y). During the training phase, the NLP algorithm learns a function f_θ that maps x to y, i.e., a transformer. This function should minimize the discrepancy between y and $f_\theta(x)$. The function f_θ depends on a set of tunable parameters, θ, which defines the architecture and the internal representation of the transformer. Given an unseen test sequence, which was not used for training, the predicted alignment is obtained by applying f_θ to this input.

Note that the values of the pairs (x, y) depend on an evolutionary model ψ that includes the topology and branch lengths of the phylogenetic tree, as well as the substitution and indel model parameters. Changing ψ would alter the training data distribution, which in turn, should lead to a different f_θ. It is possible to train the model on data from a specific ψ, or train a model in which ψ is sampled from a prior distribution over its model parameters.

If we apply the "concat" encoding with the "pairs" decoding (see Comparing Different Representations), the source dictionary size is the set of five characters and the size of the target dictionary is the set of 24 characters discussed above. In NLP translation, each word in each dictionary has to be embedded in a higher dimensional space. Thus, two different embeddings are needed (for the source and target dictionaries). In contrast, we can consider a joint dictionary for applying the "concat" or "crisscross" (see Comparing Different Representations) encodings with the "spaces" decoding. The joint dictionary of "concat" and "spaces" has six words: "A", "C", "G", "T", "–", "|". We note that the word "–" is never found in the input, while the word "|" is never found in the output. Nevertheless, we assume a joint dictionary for both the input and the output. The joint dictionary of "crisscross" "spaces" has five words: "A", "C", "G", "T", "–". Thus, the same embedding function is used for both the source language, i.e., the unaligned sequences, and the target language, the aligned sequences. One of the advantages of the joint dictionaries explained above over the separate dictionaries of "concat" with "pairs", is that the dictionaries have a fixed size regardless of how many sequences are included in the alignment (in the "pairs" representation, we should use triplets for alignments with three sequences, quadruplets for alignments with four sequences, etc.) Thus, we only use the "spaces" language when we generalize from pairwise to multiple alignment.

2.2 Splitting the Unaligned Sequences to Segments

The transformers we applied for aligning sequences are limited to sentences of up to 1,024 tokens i.e., words, which usually suffices for natural languages. When aligning biological sequences, the sentence that represents the unaligned sequences is often much longer than this threshold, and the transformers fail to translate it due to memory and run-time limitations (both these factors are proportional to the square of the sequence length). We will exemplify our approach for aligning long sequences on an example comprising five sequences, each of which of length of at least 2,000 characters. In our "segmentation" methodology, we first define a parameter that we call segment size, which depends on the number of sequences. In our example of five sequences, we selected a segment size of 110 characters. Thus, when we start the alignment process, we first translate 554 words (110 characters from each sequence and four "|" characters, when using the "concat" representation).

When aligning segments, we use dedicated transformers for this task that we call "segmented transformers". The segmented transformers share the same architecture with the standard transformers, but they are trained on different data. In addition, while standard transformers were designed to always align the entire input sentence, segmented transformers align only prefixes of the unaligned sequences. Specifically, the segmented transformers were designed so that the last column in the resulting aligned sequences does not include gaps.

To generate the training data for the segmented transformers, we simulated long alignments (see section Datasets). This provides us with long unaligned sequences, which we call X, and "true"

MSA, which we call Y. The training data for the segmented transformer are pairs of (x, y), in which x represents a segment of the unaligned sequence taken from X, and y represents a subregion of Y that corresponds to x. Both x and y are short enough to allow training a transformer.

To divide X and Y to multiple short (x, y) pairs, we repeated the following process. We first extract the first 110 characters from each of the five sequences, which will be the unaligned segment, i.e., the x. In the true alignment, we find the column with the highest index so that the number of non-gap characters of each sequence to the left of this column is smaller or equal to the segment size. Once this column is found, we inspect if it has any indels. If so, we inspect the previous column. We repeat this process until we find a column without indels. The obtained (first) column without indel characters marks the "true" alignment for this segment, i.e., the y. Hence, the source sentence is the first 110 characters of each unaligned sequence (x) and the target is the "true" alignment (y) from the first column until the indel-free column that was found. From both X and Y, we remove all the characters that appear in y (the characters that were successfully aligned). This process is repeated with the new X and Y until they are empty, i.e., they were divided to multiple (x, y) pairs. Of note, when analyzing the last segments, it is possible that fewer than 554 words are translated.

For explanatory purposes, we assume a segment size of three and five input sequences (Fig. 2). Simply diving X and Y to pairs of arbitrary length introduces various problems as illustrated in Problems Arise When Simply Splitting MSAs section.

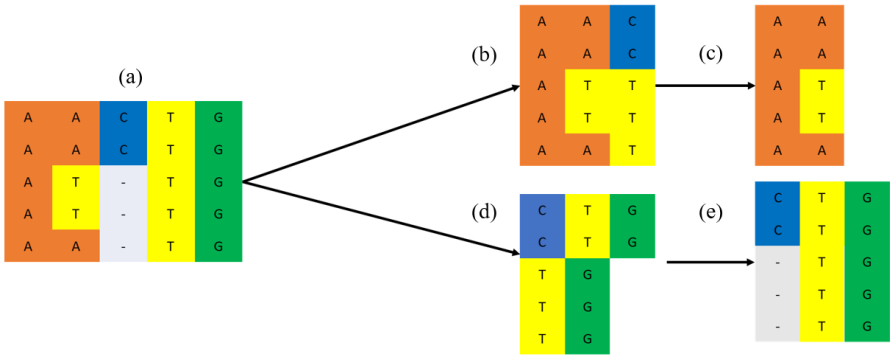

Figure 2: Creating training data for segmented transformers, explained using a segment size of three characters. (a) the true alignment; (b) The unaligned sequences of the first segment, i.e., the first x as described in the Methods section; (c) The segment of the true MSA that corresponds to the x presented in b, i.e., the first y. Note that the third column of the true MSA was not selected as it contains gaps, and the forth column was not selected, as it includes four characters of some of the original sequences (i.e., it is longer than the segment size of three); (d) The unaligned sequences of the second segment; (e) The corresponding segmented true alignment.

The above procedure described the generation of training data, for which the true alignment is known. We now turn to describe how to run a trained model. Here, we first translate the first segment. However, in order to decide which characters to remove and which to retain, we cannot rely on the identification of an indel-free column in the true alignment as we did for the training data (since we do not know the true alignment). Thus, we analyze the resulting alignment, and mark which characters of the input segments were translated from each sequence. These characters are removed, and the process is repeated. This iteration stops when all characters from the unaligned sequences were translated. Of note, different trained transformers may divide the segments differently. After aligning all of the segments, we reconstruct the resulting alignment by concatenating the resulting alignment of each segment one after another. Coverage and accuracy are measured similarly (as explained for the shorter alignments) after the concatenation.

2.3 Problems Arise When Simply Splitting MSAs

The logic for searching for an indel-free column is illustrated in Fig. 3, i.e., an example to show the problems with not indel-free last column. For explanatory purposes, we assume a segment size of three. The algorithm starts by translating the first three characters from each of the five sequences

(Fig. 3b). The indel in the last column of the first segment (Fig. 3c) introduces new complexity, as this indel has a perfect match in the next segment (Fig. 3e). In our approach, this is solved by training segmented transformers. If we choose to use indels in the last column, we move the complexity of the segment approach into the concatenation of the segments afterward. One will need to make sure the indels at the resulted alignment make sense. Of note, this approach makes each of the indels appear in only one segment (but several indels may be in the same segment). Similar cases could happen when the indel splits between two segments.

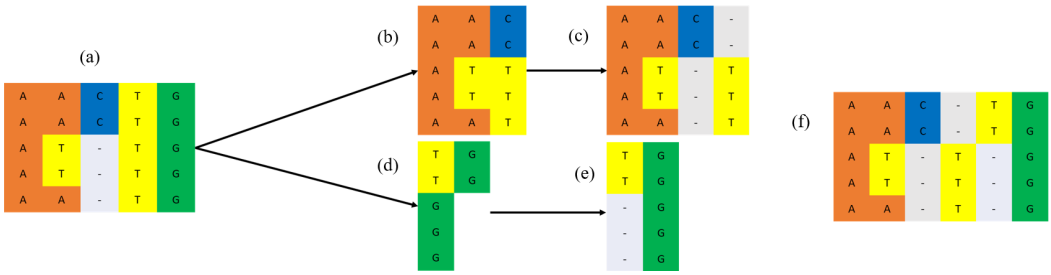

Figure 3: Importance of indel-free last column of segments. Consider (a) to be the true alignment, with a segment size of three characters. (b), (c) refers to the unaligned sequences and the aligned results, of the first segment respectively. Similarly, (d), (e) refer to the unaligned sequences and the aligned results of the second segment, respectively. (f) is the concatenation of the results. Note the difference between (a) and (f) as the third and fourth columns at (f) should be the fourth column in (a).

2.4 Comparing Different Representations

We examined several source and target languages for the translation task. In addition to "concat" (see Main Text) we consider another possibility for encoding the input sequences, and we call the corresponding language "crisscross". In this language, the first word is the first character of the first sequence and the second word is the first character of the second sequence. From the perspective of a transformer, it helps because it needs to learn closer rather than longer dependencies, compared to "concat". As the input sequences are not necessarily of the same length, a gap character ("–") is placed if no more characters are available for one of the sequences. For example, for the example input sequences in Fig. 1 ("AAG" and "ACGG"), the input sentence would be "A A A C G G - G". Thus, the number of words is always twice the length of the longest sequence.

In addition to "spaces", we additionally consider an alternative decoding for the output alignments, with the corresponding language called "pairs". We note that each column of the output alignment can be one of the following: "AA", "AC", . . . , "A-", "CA", . . . , "T-", "-A", . . . , "-T". In "pairs", each such possibility is a word. Thus, for nucleotide sequences, the dictionary size is 24. For protein sequences, the dictionary size is 440. Note that "–" is an illegal word.

We encoded the same datasets with different representations: (1) "concat" and "pairs"; (2) "concat" and "spaces" (see Introduction); (3) "crisscross" and "spaces". Then, we trained two transformers on each of the datasets, e.g., two transformers on the "concat" and "pairs", and two different transformers on "concat" and "spaces". The transformers were trained with the same tunable parameters. We compared the transformers' performance on the same test data. This comparison was done on dataset PD2 (see Table 2 in the Supplementary Text section). Note, that we did not use the "pairs" representations when shifting from pairwise to multiply alignment because the dictionary of pairwise and multiple alignments are different, which disenables transfer learning.

3 Results

3.1 Comparing Performance to Different Aligners

We compared the performance of BetaAlign to the state-of-the-art alignment algorithms: ClustalW (Larkin et al., 2007), DIALIGN (Morgenstern, 2004), MAFFT (Katoh & Standley, 2013), T-Coffee

(Notredame et al., 2000), PRANK (Löytynoja & Goldman, 2008), and MUSCLE (Edgar, 2004). For each number of sequences from two to ten, the performance was compared on a simulated test dataset comprising 3,000 nucleotide MSAs (Fig. 4a). BetaAlign had the lowest error rate for any number of aligned sequences (paired t-test; $P < 10^{(-20)}$). For all alignment methods, the error increases as the number of sequences increases. Notably, ClustalW and DIALIGN were much more affected by the increase of the number of input sequences, compared to other methods. Of note, this analysis was conducted by applying the "concat" source language with the "spaces" target language.

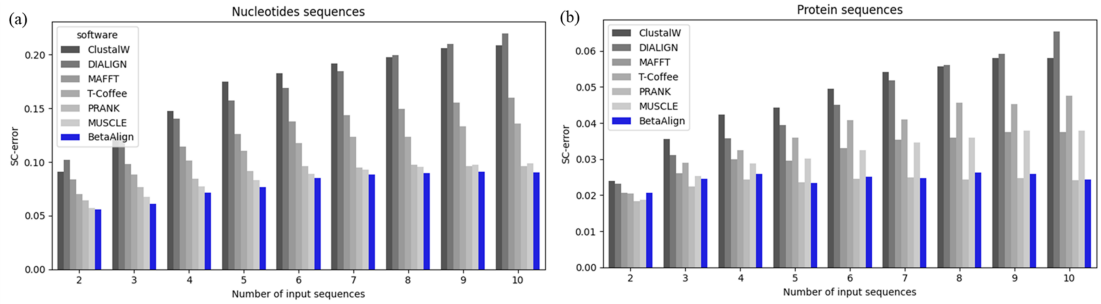

Figure 4: Comparing the accuracy of inferred alignments of ClustalW, DIALIGN, MAFFT, T-Coffee, PRANK, MUSCLE, and BetaAlign. Nucleotide (a) and protein (b) sequences were analyzed on datasets SND1 and SPD1LG, respectively. The x-axis is the number of input sequences, and the y-axis is the SC-error, which quantifies accuracy (see Supplementary Information).

We repeated the same analysis for the protein dataset. The error of all methods was low compared to nucleotide sequences, most likely reflecting the increase in alphabet size, which helps identify homologous regions (Fig. 4b). PRANK and BetaAlign outperformed all other aligners. As seen from Fig. 4, there is no single optimal method to align all types of sequences.

3.2 Generalizing to Large MSAs

Aligning long sequences with transformers introduces new challenges. The attention map calculated inside the transformer is $\emptyset(n^2m^2)$, both in term of memory and running time, where n and m are the number of sequences and the sequences' length, respectively. Note, the mn reflects the total words that need to be translated. Thus, increasing the alignment size causes memory and time problems. This is a known problem in the NLP domain, and so different efficient transformers were created in order to tackle this issue (Lin et al., 2021). Those architectures cannot be directly applied to the alignment problem (see the Attention Map Comparison section). Hence, we developed a new way to align longer sequences by splitting the input sequences into short segments and training the transformers to align each segment (the size of the segments varies along the run, but is approximately 100 alignment columns). We compared the performance of all aligners on MSAs of five sequences, with true alignment lengths up to 2,500 bases and 2,508 amino acids, for nucleotides and protein sequences, respectively. Clearly, the performance of BetaAlign is affected by the division of sequences into short segments, both when aligning nucleotide (Fig. 5a) and protein (Fig. 5b) sequences. Notably, even with this approximation, BetaAlign is more accurate than the widely used aligners MAFFT and ClustalW for nucleotide sequences and more than MAFFT, ClustalW, and MUSCLE for protein sequences.

3.3 Model Misspecification

In all the above analyses all positions within a specific MSA were generated by a model controlled by a specific set of parameters, in other words, there was no spatial model variation. We next evaluated how sensitive is BetaAlign to model misspecification. Specifically, we trained BetaAlign on alignments without spatial variations, but tested its performance on alignments generated with spatial variations. Test datasets were created by generating four "domains", each with a different indel dynamics (see Supplementary Methods). BetaAlign was substantially more accurate on the test protein dataset compared to other aligners (Fig. 6a). On the nucleotide dataset, BetaAlign SC-error was lower than two aligners but higher than the other two (Fig. 6b). We note that without model

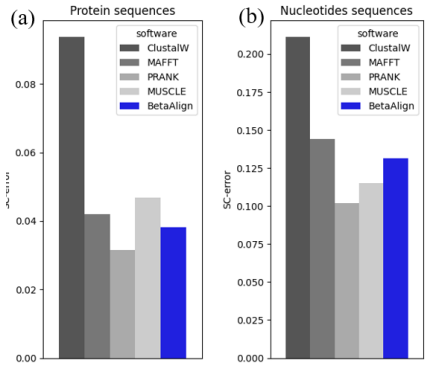

Figure 5: Running on large MSAs. Comparing the results of different aligners on longer sequences for five sequences for nucleotide (a) and protein (b) sequences. The average MSA length was 760 bases and 765 amino acids, for nucleotide and protein sequences, respectively. These datasets were aligned after dividing the input into short segments, thus overcoming computational memory and run time limitations introduced by large MSAs.

misspecification, BetaAlign had the highest accuracy compared to all other aligners on nucleotide datasets and was not the best for protein datasets. Model misspecification changed this order, as BetaAlign is now the most accurate aligner for protein dataset and had intermediate accuracy on nucleotide dataset. In general, machine-learning models are sensitive to deviations between the training and testing distributions. We found it surprising as in this case, the classic aligners on the protein dataset were more sensitive than the BetaAlign.

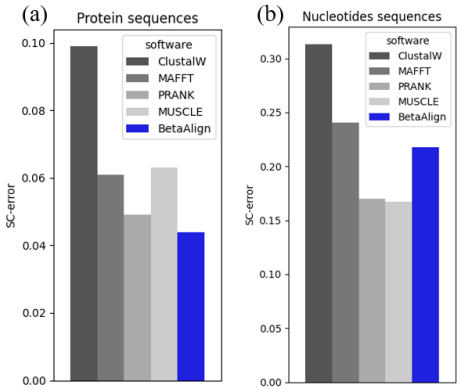

Figure 6: The effect of model misspecification on the accuracy of protein (a) and nucleotide (b) datasets. Train and test alignments had 10 sequences. Spatial variation was only introduced to the test data, i.e., different regions of the alignment were simulated with different sets of indel model parameters.

3.4 Testing Alternative Representations Schemes

Table 1 displays the performance of transformers trained with various encodings for pairwise protein alignment (as explained in Materials and Methods). All six variants had similar accuracy, as measured by the SC-score. The "pairs" target language resulted in the highest coverage. For the task of multiple sequence alignments, we decided not to work with the generalization of the "pairs" target language because in this case the size of the dictionary increases exponentially with the number of input sequences (if we denote n as the number of input sequences, the output protein dictionary size is $22^n - 1$). Furthermore, because the dictionary size depends on n, the "pairs" target language hinders using transfer learning. Note that the "spaces" and "crisscross" dictionaries do not change when increasing the number of input sequences. As the performance using the "concat" source lan-

Table 1: Transformers performance trained on pairwise protein alignments with source and target languages (dataset PD2). Each combination of source and target language was tested with two values for the max-token parameter, resulting in two transformers: "original" and "alternative".

TRANSFORMER NAME	SOURCE LANGUAGE	TARGET LANGUAGE	MAX TOKENS	SC-SCORE	COVERAGE
original	"concat"	"pairs"	4096	0.997	0.939
alternative	"concat"	"pairs"	2048	0.997	0.952
original	"concat"	"spaces"	4096	0.997	0.835
alternative	"concat"	"spaces"	2048	0.998	0.635
original	"crisscross"	"spaces"	4096	0.991	0.503
alternative	"crisscross"	"spaces"	2048	0.995	0.636

guage was slightly better than that of the "crisscross" source language, the former was applied for all analyses.

3.5 Attention Map Comparison

BetaAlign harnesses NLP-based translation methods to analyze biological data. However, it is clear that biological data are substantially different from natural languages data, e.g., they have a much smaller dictionary size (six for nucleotides using the "spaces" representation, compared to an order of a hundred thousand words in vocabularies of natural languages). Attention maps provide a gateway for understanding the internal properties of various transformers (explained more below). Put simply, attention maps show on which part of the sequences the transformer focuses while generating the next word. Fig. 7a displays the averaged attention map gathered from the attention layers of the decoder (in this case, when processing an amino-acid MSA of seven sequences, from PD14). Fig. 7b shows an attention map from an NLP English to French translation task (Belinkov & Glass, 2019; Bahdanau et al., 2016).

Note that each attention map corresponds to translation of a single sentence. In the case of Fig. 7b, the translation is of the sentence "The agreement of the European Economic Area was signed in August 1992.". The sentence translated in Fig. 7a is a concatenation of seven unaligned sequences of length 40 nucleotides. Clearly, natural language sentences are much shorter, on average, than sentences translated by BetaAlign, and thus, the BetaAlign attention map is much larger (consisting of hundreds of words) than the natural language one (consisting of dozens of words). Moreover, the attention in the natural languages is focused on one diagonal while the attention in BetaAlign is focused on several diagonals, the number of which equals the number of inputs sequences. The difference stems from the fact that in the BetaAlign translation task, generating the next word requires assigning more weight to information that is distributed among all sequences. Of note, this analysis was conducted on the "concat" and "spaces" for the source and target languages, respectively.

Fig. 7c is an enlargement of the attention map around the rightmost diagonal of Fig. 7a. The x-axis refers to the characters of the input sentence and the y-axis refers to the translated characters of the output sentence. As can be seen, the diagonal is actually a step of seven rows and one column. This is the logical explanation as each of the seven rows extracts information from this column, i.e., the characters of a column at the resulting MSA are all dependent on the same character in the unaligned sequences. The red arrows mark the corresponding output and input characters. As can be seen, the weight is higher in this area, i.e., the transformer learned the importance of the corresponding area.

Due to the dynamics of our attention maps (specifically, the focus split), we cannot easily use NLP architectures for longer sentences, which have dedicated attention maps (Lin et al., 2021). In our segmentation strategy, instead of calculating the entire attention map, we focus the attention on specific places of the diagonals. Future research will focus on improving attention calculations.

4 Discussion

We have designed and implemented a new technology that aligns sequences with accuracy that competes with state-of-the-art methods. Our deep learning-based approach is fundamentally different from current algorithms, which rely on conventional scoring schemes. Our method allows us to personalize the alignment process for inputs that differ in their evolutionary dynamics. The

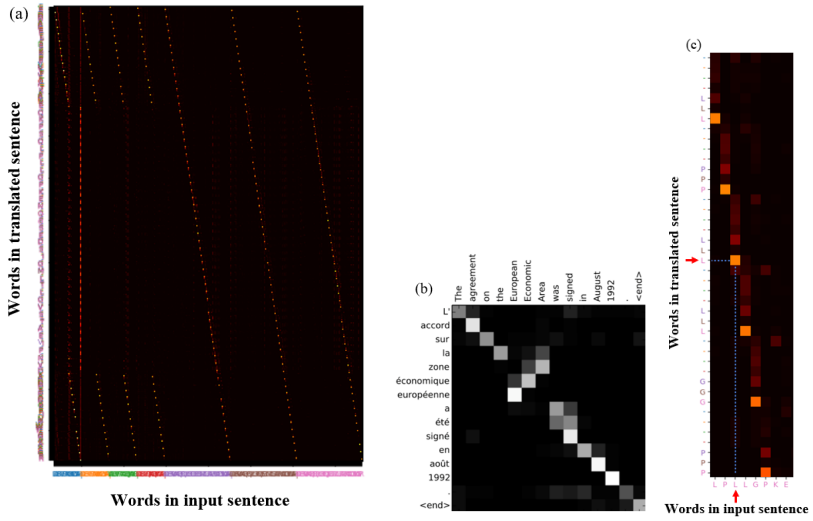

Figure 7: Comparing the attention maps of BetaAlign and other NLP tasks. The x-axis is the input words, and the y-axis is the output words. The attention splits between all the input words when generating the next word. (a) attention map extracted from the last layer of the BetaAlign's decoder. One can see that the NLP task (b) refers to only one location of the input sentence while BetaAlign refers to multiply locations, one for each of the input sequences. To visualize this, we colored each of the unaligned sequences in a different color as well as the amino acid that refers to this unaligned sequence. When zooming into the map, there is a step of the attention of the four left diagonals, that we identified as an indel. In this indel, the first "|" have a meaningful part. (c) zooming in to one of the diagonals of (a). The red arrows represent the corresponding output and input characters "L".

increased knowledge regarding the differences in mutation processes and selection regimes among different species allows us to easily train BetaAlign on clade-specific datasets, i.e., we are capable of creating different aligners per kingdom and even per species. Similarly, BetaAlign can be trained to capture the evolutionary dynamics of different regions within the genome, e.g., introns versus promotors versus other non-coding regions. Moreover, this method has the potential to capture variations among different types of protein-coding genes, e.g., transmembrane versus cytoplasmatic proteins.

One can argue that BetaAlign heavily depends on the simulator. In other words, if the simulator does not reflect biological realism, the resulting aligner may produce inaccurate MSAs for empirical data. However, the design of BetaAlign allows switching between simulators easily. This allows research groups to focus on mimicking the sequence evolutionary process to create better simulators, which in turn, will improve the alignment accuracy. Since tuning a simulator is an easier task than creating a new alignment method, we anticipate a shift in the scientific community from developing better aligners to developing more realistic simulation schemes. BetaAlign can learn those specific alignment rules without the need to specifically provide them as an input to the program. This feature abolishes the need to provide any assumptions regarding the evolutionary process at the development stage of the aligner. For example, if a research group assumes a translocation or recombination happened between proteins of different species, they can create a simulator reflecting this process and train BetaAlign to align those sequences, instead of developing new methods to consider translocation or recombination (to the best of our knowledge, current aligners do not consider these types of events).

In this pioneering paper, we have demonstrated the power of applying NLP tools to biological questions. We believe that this link is essential and will change many aspects of computational biology. Leveraging deep-learning models discovers new research options with promising results. Of course, the NLP methods could be used in many other biological tasks, e.g., one could translate unaligned sequences to generate the ancestor sequence or classify different protein sequences to their corresponding GO annotations. Many downstream tasks could benefit from accurate features which could be extracted by embedding the genomic data.

REFERENCES

Federico Abascal, David Posada, and Rafael Zardoya. MtArt: A New Model of Amino Acid Replacement for Arthropoda. *Molecular Biology and Evolution*, 24(1):1–5, January 2007. ISSN 0737-4038. doi: 10.1093/molbev/msl136. URL https://doi.org/10.1093/molbev/msl136.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016. URL http://arxiv.org/abs/1409.0473. arXiv: 1409.0473.

Gunjan Baid, Daniel E. Cook, Kishwar Shafin, Taedong Yun, Felipe Llinares-López, Quentin Berthet, Aaron M. Wenger, William J. Rowell, Maria Nattestad, Howard Yang, Alexey Kolesnikov, Armin Töpfer, Waleed Ammar, Jean-Philippe Vert, Ashish Vaswani, Cory Y. McLean, Pi-Chuan Chang, and Andrew Carroll. DeepConsensus: Gap-Aware Sequence Transformers for Sequence Correction. preprint, Bioinformatics, August 2021. URL http://biorxiv.org/lookup/doi/10.1101/2021.08.31.458403.

Yonatan Belinkov and James Glass. Analysis Methods in Neural Language Processing: A Survey. *Transactions of the Association for Computational Linguistics*, 7:49–72, April 2019. ISSN 2307-387X. doi: 10.1162/tacl_a_00254. URL https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00254/43503/Analysis-Methods-in-Neural-Language-Processing-A.

Jia-Ming Chang, Paolo Di Tommaso, and Cedric Notredame. TCS: a new multiple sequence alignment reliability measure to estimate alignment accuracy and improve phylogenetic tree reconstruction. *Molecular Biology and Evolution*, 31(6):1625–1637, June 2014. ISSN 1537-1719. doi: 10.1093/molbev/msu117.

R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, March 2004. ISSN 1362-4962. doi: 10.1093/nar/gkh340. URL https://academic.oup.com/nar/article-lookup/doi/10.1093/nar/gkh340.

S Henikoff and J G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(22):10915–10919, November 1992. ISSN 0027-8424. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC50453/.

Brian L. Hill, Nadav Rakocz, Ákos Rudas, Jeffrey N. Chiang, Sidong Wang, Ira Hofer, Maxime Cannesson, and Eran Halperin. Imputation of the continuous arterial line blood pressure waveform from non-invasive measurements using deep learning. *Scientific Reports*, 11(1):15755, December 2021. ISSN 2045-2322. doi: 10.1038/s41598-021-94913-y. URL https://www.nature.com/articles/s41598-021-94913-y.

Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *ARTIFICIAL INTELLIGENCE*, pp. 7.

D. T. Jones, W. R. Taylor, and J. M. Thornton. The rapid generation of mutation data matrices from protein sequences. *Computer applications in the biosciences: CABIOS*, 8(3):275–282, June 1992. ISSN 0266-7061. doi: 10.1093/bioinformatics/8.3.275.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-021-03819-2. URL https://www.nature.com/articles/s41586-021-03819-2.

Kazutaka Katoh and Daron M. Standley. MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability. *Molecular Biology and Evolution*, 30(4):772–780, April 2013. ISSN 0737-4038. doi: 10.1093/molbev/mst010. URL https://doi.org/10.1093/molbev/mst010.

Carsten Kemena and Cedric Notredame. Upcoming challenges for multiple sequence alignment methods in the high-throughput era. *Bioinformatics (Oxford, England)*, 25(19):2455–2465, October 2009. ISSN 1367-4811. doi: 10.1093/bioinformatics/btp452.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. December 2014. doi: 10.48550/arXiv.1412.6980. URL https://arxiv.org/abs/1412.6980v9.

M.A. Larkin, G. Blackshields, N.P. Brown, R. Chenna, P.A. McGettigan, H. McWilliam, F. Valentin, I.M. Wallace, A. Wilm, R. Lopez, J.D. Thompson, T.J. Gibson, and D.G. Higgins. Clustal W and Clustal X version 2.0. *Bioinformatics*, 23(21):2947–2948, November 2007. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btm404. URL https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btm404.

S. Q. Le and O. Gascuel. An Improved General Amino Acid Replacement Matrix. *Molecular Biology and Evolution*, 25(7):1307–1320, April 2008. ISSN 0737-4038, 1537-1719. doi: 10.1093/molbev/msn067. URL https://academic.oup.com/mbe/article-lookup/doi/10.1093/molbev/msn067.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14539. URL http://www.nature.com/articles/nature14539.

Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A Survey of Transformers. *arXiv:2106.04554 [cs]*, June 2021. URL http://arxiv.org/abs/2106.04554. arXiv: 2106.04554.

Felipe Llinares-López, Quentin Berthet, Mathieu Blondel, Olivier Teboul, and Jean-Philippe Vert. Deep embedding and alignment of protein sequences. preprint, Bioinformatics, November 2021. URL http://biorxiv.org/lookup/doi/10.1101/2021.11.15.468653.

Gil Loewenthal, Dana Rapoport, Oren Avram, Asher Moshe, Elya Wygoda, Alon Itzkovitch, Omer Israeli, Dana Azouri, Reed A Cartwright, Itay Mayrose, and Tal Pupko. A Probabilistic Model for Indel Evolution: Differentiating Insertions from Deletions. *Molecular Biology and Evolution*, 38(12):5769–5781, December 2021. ISSN 1537-1719. doi: 10.1093/molbev/msab266. URL https://doi.org/10.1093/molbev/msab266.

Ari Löytynoja. Phylogeny-aware alignment with PRANK. *Methods in Molecular Biology (Clifton, N.J.)*, 1079:155–170, 2014. ISSN 1940-6029. doi: 10.1007/978-1-62703-646-7_10.

Ari Löytynoja and Nick Goldman. Phylogeny-Aware Gap Placement Prevents Errors in Sequence Alignment and Evolutionary Analysis. *Science*, 320(5883):1632–1635, June 2008. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.1158395. URL https://www.science.org/doi/10.1126/science.1158395.

B. Morgenstern. DIALIGN: multiple DNA and protein sequence alignment at BiBiServ. *Nucleic Acids Research*, 32(Web Server):W33–W36, July 2004. ISSN 0305-1048, 1362-4962. doi: 10.1093/nar/gkh373. URL https://academic.oup.com/nar/article-lookup/doi/10.1093/nar/gkh373.

David A. Morrison. Why Would Phylogeneticists Ignore Computerized Sequence Alignment? *Systematic Biology*, 58(1):150–158, February 2009. ISSN 1076-836X, 1063-5157. doi: 10.1093/sysbio/syp009. URL https://academic.oup.com/sysbio/article/58/1/150/1675067.

S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970. ISSN 0022-2836. doi: 10.1016/0022-2836(70)90057-4.

Cédric Notredame, Desmond G Higgins, and Jaap Heringa. T-coffee: a novel method for fast and accurate multiple sequence alignment 1 1Edited by J. Thornton. *Journal of Molecular Biology*, 302(1):205–217, September 2000. ISSN 00222836. doi: 10.1006/jmbi.2000.4042. URL https://linkinghub.elsevier.com/retrieve/pii/S0022283600940427.

T Heath Ogden and Michael S Rosenberg. Multiple Sequence Alignment Accuracy and Phylogenetic Inference. *Systematic Biology*, 55(2):314–328, April 2006. ISSN 1063-5157. doi: 10.1080/10635150500541730. URL https://doi.org/10.1080/10635150500541730.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A Fast, Extensible Toolkit for Sequence Modeling, April 2019. URL http://arxiv.org/abs/1904.01038. arXiv:1904.01038 [cs].

O. Penn, E. Privman, G. Landan, D. Graur, and T. Pupko. An Alignment Confidence Score Capturing Robustness to Guide Tree Uncertainty. *Molecular Biology and Evolution*, 27(8):1759–1767, August 2010. ISSN 0737-4038, 1537-1719. doi: 10.1093/molbev/msq066. URL https://academic.oup.com/mbe/article-lookup/doi/10.1093/molbev/msq066.

G. Pesole, C. Gissi, A. De Chirico, and C. Saccone. Nucleotide substitution rate of mammalian mitochondrial genomes. *Journal of Molecular Evolution*, 48(4):427–434, April 1999. ISSN 0022-2844. doi: 10.1007/pl00006487.

Eyal Privman, Osnat Penn, and Tal Pupko. Improving the performance of positive selection inference by filtering unreliable alignment regions. *Molecular Biology and Evolution*, 29(1):1–5, January 2012. ISSN 1537-1719. doi: 10.1093/molbev/msr177.

Tal Pupko and Itay Mayrose. A gentle Introduction to Probabilistic Evolutionary Models. pp. 22, 2020.

Nadav Rakocz, Jeffrey N. Chiang, Muneeswar G. Nittala, Giulia Corradetti, Liran Tiosano, Swetha Velaga, Michael Thompson, Brian L. Hill, Sriram Sankararaman, Jonathan L. Haines, Margaret A. Pericak-Vance, Dwight Stambolian, Srinivas R. Sadda, and Eran Halperin. Automated identification of clinical features from sparsely annotated 3-dimensional medical imaging. *npj Digital Medicine*, 4(1):44, December 2021. ISSN 2398-6352. doi: 10.1038/s41746-021-00411-w. URL http://www.nature.com/articles/s41746-021-00411-w.

Álvaro Rubio-Largo, Leonardo Vanneschi, Mauro Castelli, and Miguel A. Vega-Rodríguez. Swarm intelligence for optimizing the parameters of multiple sequence aligners. *Swarm and Evolutionary Computation*, 42:16–28, October 2018. ISSN 22106502. doi: 10.1016/j.swevo.2018.04.003. URL https://linkinghub.elsevier.com/retrieve/pii/S2210650216304667.

Itamar Sela, Haim Ashkenazy, Kazutaka Katoh, and Tal Pupko. GUIDANCE2: accurate detection of unreliable alignment regions accounting for the uncertainty of multiple parameters. *Nucleic Acids Research*, 43(W1):W7–W14, July 2015. ISSN 0305-1048, 1362-4962. doi: 10.1093/nar/gkv318. URL https://academic.oup.com/nar/article-lookup/doi/10.1093/nar/gkv318.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. ISSN 1533-7928. URL http://jmlr.org/papers/v15/srivastava14a.html.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks, December 2014. URL http://arxiv.org/abs/1409.3215. arXiv:1409.3215 [cs].

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision, December 2015. URL http://arxiv.org/abs/1512.00567. arXiv:1512.00567 [cs].

J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, November 1994. ISSN 0305-1048. doi: 10.1093/nar/22.22.4673.

Richard Van Noorden, Brendan Maher, and Regina Nuzzo. The top 100 papers. *Nature*, 514(7524): 550–553, October 2014. ISSN 1476-4687. doi: 10.1038/514550a.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017. URL http://arxiv.org/abs/1706.03762. arXiv:1706.03762 [cs].

Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*, 2018:7068349, 2018. ISSN 1687-5273. doi: 10.1155/2018/7068349.

Huai-Chun Wang, Karen Li, Edward Susko, and Andrew J Roger. A class frequency mixture model that adjusts for site-specific amino acid frequencies and improves inference of protein phylogeny. *BMC Evolutionary Biology*, 8(1):331, December 2008. ISSN 1471-2148. doi: 10.1186/1471-2148-8-331. URL https://bmcevolbiol.biomedcentral.com/articles/10.1186/1471-2148-8-331.

L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, 1(4):337–348, 1994. ISSN 1066-5277. doi: 10.1089/cmb.1994.1.337.

Martin Wu, Sourav Chatterji, and Jonathan A. Eisen. Accounting For Alignment Uncertainty in Phylogenomics. *PLoS ONE*, 7(1):e30288, January 2012. ISSN 1932-6203. doi: 10.1371/journal.pone.0030288. URL https://dx.plos.org/10.1371/journal.pone.0030288.

Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent Trends in Deep Learning Based Natural Language Processing, November 2018. URL http://arxiv.org/abs/1708.02709. arXiv:1708.02709 [cs].

A SUPPLEMENTARY INFORMATION

A.1 SUPPLEMENTARY METHODS

A.1.1 PREVIOUS APPROACHES FOR ALIGNING SEQUENCES AND MOTIVATION FOR THE CURRENT ALGORITHM

The Needleman–Wunsch algorithm was the first to use dynamic programming to efficiently find the best global scoring alignment among two sequences (Needleman & Wunsch, 1970). MSA inference was shown to be an NP-complete problem (Wang & Jiang, 1994), making the inference task impractical for a large set of sequences. To overcome this hurdle, popular MSA algorithms, such as MAFFT (Katoh & Standley, 2013) and PRANK (Löytynoja, 2014), use heuristics to reduce the search space and consequently, the running time.

There is extensive knowledge regarding the variability of the evolutionary process among different datasets and lineages. For example, amino-acid replacement matrices vary between proteins encoded in the nuclear genome, the mitochondria, and plastids (Pesole et al., 1999). Indel dynamics also highly vary between datasets and among different phylogenetic groups (Loewenthal et al., 2021). Site-specific evolutionary rates also vary along the analyzed sequence, for example, amino-acid sites that are exposed to the solvent tend to have higher evolutionary rates compared to buried sites (Wang et al., 2008). Alignment algorithms using default configurations implicitly assume that the evolutionary dynamics do not vary among different datasets and within a single dataset. The general inability of MSA inference algorithms to automatically tune their scoring scheme to the specific dataset being analyzed is a shortcoming of present alignment programs. This "one matrix fits all biological datasets" assumption employed by current methodologies raises fundamental questions about the correctness of alignments produced by such methods.

One of the key concepts in learning algorithms, in general, and in deep-learning algorithms in particular, is the ability to learn from previously annotated data, i.e., to generalize from previous observations to unseen cases. For the task of alignment inference, a deep-learning algorithm should learn from "true" alignments (e.g., simulated sequences for which the correct alignment is known) and apply the obtained knowledge to align novel sequences. In this work, we demonstrate the use of sequence-to-sequence learning to align biological sequences. Our main assumption is that by utilizing NLP models we can learn to capture the evolutionary dynamics of biological sequences.

A.1.2 GENERATING TRUE MSAS USING SPARTAABC

We used SpartaABC (Loewenthal et al., 2021) to generate millions of true alignments. SpartaABC requires the following input: (1) a rooted phylogenetic tree, which includes a topology and branch lengths; (2) a substitution model (amino acids or nucleotides); (3) root sequence length; (4) the indel model parameters, which include: insertion rate (R_I), deletion rate (R_D), a parameter for the insertion Zipfian distribution (A_I), and a parameter for the deletion Zipfian distribution (A_D). MSAs were simulated along random phylogenetic tree topologies generated using the program ETE version 3.0 27 with default parameters.

We describe the data created for Fig. 4 in the main text. We generated 395,000 and 3,000 protein MSAs i.e., SPD1, with ten sequences that were used as training and testing data, respectively. We generated the same number of DNA MSAs i.e., SND1. Each such MSA was simulated along a different random phylogenetic tree. The random topologies were generated using the program ETE version 3.0 27 with default parameters. For each tree, branch lengths were drawn from a uniform distribution in the range $(0.5, 1.0)$. Next, the sequences were generated using SpartaABC with the following parameters: $R_I, R_D \in (0.0, 0.05)$, $A_I, A_D \in (1.01, 2.0)$. The alignment lengths as well as the sequence lengths of the tree leaves vary within and among datasets as they depend on the indel dynamics and the root length. The root length was sampled uniformly in the range $[32, 44]$. Unless stated otherwise, all protein datasets were generated with the WAG+G model, and all DNA datasets were generated with the GTR+G model, with the following parameters: (1) frequencies for the different nucleotides $(0.37, 0.166, 0.307, 0.158)$, in the order "T", "C", "A" and "G"; (2) with the substitutions rate $(0.444, 0.0843, 0.116, 0.107, 0.00027)$, in the order "a", "b", "c", "d", and "e" for the substitution matrix (Fig. 8).

$$
\begin{array}{cc}
 & \begin{array}{cccc} \multicolumn{4}{c}{To} \\ T & C & A & G \end{array} \\
Q = & \begin{pmatrix} . & a\pi_C & b\pi_A & c\pi_G \\ a\pi_T & . & d\pi_A & e\pi_G \\ b\pi_T & d\pi_C & . & f\pi_G \\ c\pi_T & e\pi_C & f\pi_A & . \end{pmatrix} \begin{array}{c} T \\ C \\ A \\ G \end{array} \quad From
\end{array}
$$

Figure 8: Substitution matrix.

A.1.3 Generating Long MSAs

For the dataset used in Fig. 5, we generated 395,000 and 3,000 protein MSAs with five sequences, which were used as training and testing data, respectively. We generated the same number of DNA MSAs. As we used segments in those datasets, we divided the alignments into smaller pieces (explained below) with a fixed size of 110. Hence, for the protein dataset the number of sentences in the training data are 2,857,364. For the DNA dataset, the number of sentences in the training data are 2,424,287. The amount of testing segments is not constant as it depends on the transformer results of the previous segment. Of note, when using segments, the number of resulting sentences is higher than the number of alignments (depending on the alignment length and the segment size). In contrast, when aligning shorter sequences, each alignment is a sentence. We used the same SpartaABC parameters as were used for generating the dataset for Fig. 4 (explained above), except for a root length sampled uniformly from the interval $[480, 1, 120]$ and a branch length drawn from a uniform distribution in the range $(0.1, 0.15)$. For detailed information about processing the segments dataset, see the Splitting the Unaligned Sequences to Segments section.

A.1.4 Datasets

Various datasets were generated to train the transformers and to test the performance of BetaAlign. The description of how each data were generated and the parameters used to train each transformer are provided as Supplementary Table 2.

A.1.5 Evaluating Accuracy of Inferred Alignments

Alignment accuracy is often quantified by the column score (CS), which counts the number of alignment columns in the inferred alignment that are identical to the "true" alignment (Penn et al., 2010). We note that we request both the coordinates and the characters of each column to match. For example, consider the case that in a true pairwise alignment, position 17 of the first sequences is an "A", which is matched to an "A" in position 19 of the second sequences. If in the inferred alignment, position 17 of the first sequence matches position 21 in the second sequence, and they are both "A", this is not considered a match as the coordinates are not the same. To normalize this score to the range of $[0, 1]$, we divided the CS by the total number of columns in the true MSA. The resulting score is termed here SC-score (for the score of columns). The SC-error is defined as 1.0 – SC-score, and thus reflects the level of disagreement between the inferred and true MSAs.

A.1.6 Coverage

When the trained BetaAlign is applied to test data, the output is a sequence of words. It is possible for this sequence of words to not reflect a biologically meaningful alignment. For example, while an alignment algorithm, in general, should only introduce new gap characters, here, our algorithm sometimes wrongly mutates the characters of the input sequence or outputs an alignment in which the sequences are not of the same length (although, by alignment definition, they should be). We call such an output an "invalid alignment". Such cases are illustrated in Fig. 9. A detailed explanation about this metric is at the "Calculating Coverage" section below.

It is possible to train several transformers on the same training dataset. These transformers can have different configurations, e.g., as a result of using different optional tunable parameters. One such tunable parameter is, for example, the learning rate (see below). We train a set of transformers on the same training data. It is often the case that one transformer outputs an invalid alignment, while others output a valid alignment. We use only the valid alignments to determine the final output. The

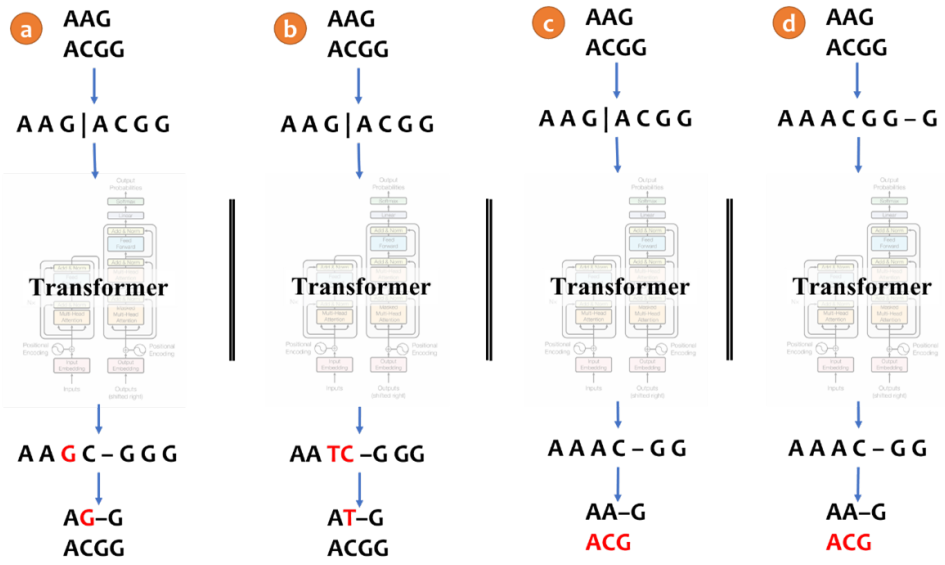

Figure 9: Invalid alignments. (a) Although the first input sequence is AAG, the first row in the resulting pairwise alignment is AG–G, which is invalid as the algorithm not only introduced gaps but also mutated a character. Here, sequences were encoded using the "concat" representation and decoded with "spaces"; (b) A similar problem may arise when decoding with "pairs"; (c) An example of an invalid pairwise alignment when applying the "concat" and "spaces" representations, stemming from an unequal length of rows in the resulting MSA. Note that this problem can never appear using "pairs"; (d) A similar problem may arise in the "crisscross" encoding. The transformer architecture illustration is adapted from (Vaswani et al., 2017).

performance of BetaAlign is always computed on the output of the original transformer. We term the percentage of cases for which we can obtain valid alignments as "coverage". The more transformers we use, the higher the "coverage", and thus the higher our confidence in the final output. Of note, using alternative transformers also provides alternative alignments, which can be used, for example, to estimate alignment uncertainty (Wu et al., 2012; Sela et al., 2015).

A.1.7 Preprocessing the Input for the Machine Translation Task

To train the NLP models, we first represent the input unaligned sequences in one of the two source languages explained above ("concat" or "crisscross"). The true MSAs were also represented in the target languages ("pairs" or "spaces"). The actual training of the NLP transformers as well as the pre-processing step were done using the Fairseq library (Ott et al., 2019), which is implemented in Python. Six text files are used as input for the pre-processing step before actually running the transformer: unaligned sequences and true MSAs, for training, validation, and testing data.

A.1.8 Preprocessing Inputs and Outputs

In addition, for running the preprocessing steps, one has to declare names for the input and output languages. Moreover, one can explicitly state if the same dictionary is used for both the source and target languages, i.e., whether a joint dictionary should be assumed. In our case, when we apply the combinations of "concat" and "spaces" or "crisscross" and "spaces", the dictionary is joint, while for the combination "concat" and "pairs", the dictionary is not joint and hence two different dictionaries are internally generated by the library (one for the source language and the other for the target language). The output of this preprocessing step is binary files that are later used as input to the transformer. This preprocess step allows running different configurations of the transformers (or potentially, other machine-learning models) on the same input files, without the need to perform the same computations on the input files over and over again.

A.1.9 TRAINING THE TRANSFORMERS

We first assessed various transformer configurations, which differ in their training parameters: max tokens, learning rate and warmup updates. Based on their performance on the pairwise nucleotide and protein alignment datasets (Supplementary Table 2), we selected two optimal transformer configurations, which we call "original" and "alternative". The learning rate and warmup values for both transformers are 5E-5 and 3,000, respectively. The max token parameter values are 4,096 and 2,048 for the original and alternative transformers, respectively.

We used label-smoothed cross entropy (Szegedy et al., 2015) to compute the loss of the model with a dropout (Srivastava et al., 2014) rate of 0.3. We used "Adam" (Kingma & Ba, 2014) as the optimizer of the model with 0.9 forgetting factors for gradients and 0.98 for the and second moments of gradients.

All model evaluations are executed on GPU machines, Tesla V100-SXM2-32GB. The Weights and Biases platform (https://wandb.ai) was used to follow the progress of the training steps and helped decide when to stop the learning processes.

A.1.10 TRANSFER LEARNING IMPLEMENTATION

In our work, transfer learning was repeatedly used for training the transformers. The first protein transformer was trained on a simple dataset of pairwise amino acid sequences (we denote this dataset "PD1", for protein dataset 1). Its weights were randomly sampled with default values of the Fairseq library. The resulting trained transformer is termed "PT1", for protein transformer 1. PT1 was next trained on PD2, resulting in PT2, etc. A similar process was used to train the nucleotide-based transformers (NT1, NT2, etc.) on nucleotide (DNA) datasets (ND1, ND2, etc.) Of note, transfer learning was used in this study only when the previous and the currently processed data are encoded using the same representations, i.e., they share the same dictionaries.

A.1.11 CALCULATING COVERAGE

Each alignment was passed through both the original and alternative transformers. A valid alignment is defined by the following rules: (1) All alignment rows are of the same length; (2) When removing all gaps from each row of the alignment, the result equals to the unaligned input sequences. For example, for the unaligned inputs: "AAG" and "ACGG", the inferred alignment, "A-AG" over "ACGG" is valid, whereas for the same input the alignment "A-AG" over "ACGC" is invalid (see also Fig. 9). The coverage is defined as the fraction of valid alignments.

A.1.12 GENERATING MISSPECIFICATION DATASET

To create misspecification datasets, we simulated alignments that have four different regions, each evolving under a different indel model, i.e., each part of the alignment has different indel parameters. The R_I and R_D are sampled from the ranges: $[0.01, 0.02)$, $[0.02, 0.03)$, $[0.03, 0.04)$, $[0.04, 0.05)$ for the first, second, third and fourth regions of the alignment, respectively. The A_I and A_D are sampled from the ranges: $[1.01, 1.2)$, $[1.2, 1.4)$, $[1.4, 1.6)$, $[1.6, 1.8)$ for the different regions, respectively. Those values were assumed both when simulating nucleotide and protein datasets. For protein sequences, each region was simulated assuming a different amino-acid replacement matrix: JTT (Jones et al., 1992), BLOSUM (Henikoff & Henikoff, 1992), mtART (Abascal et al., 2007), and LG Le & Gascuel (2008), for the first to the fourth domains, respectively. For nucleotide sequences, the GTR1, JC, GTR2, JC, were assumed for each region, respectively (Pupko & Mayrose, 2020). The GTR1 and GTR2 frequencies are (0.37, 0.166, 0.307, 0.158) and (0.265, 0.182, 0.171, 0.382) for the frequency of T, C, A, and G, respectively. The assume entries in the rate matrix were: (0.444, 0.0843, 0.116, 0.107, 0.00027) and (1.387, 1.79, 0.431, 0.321, 4.947), for a, b, c, d, and e as stated in matrix Q (see Generating True MSAs Using SpartaABC section), for GTR1 and GTR2, respectively.

A.1.13 EXTRACTING THE ATTENTION MAPS

In the Fairseq library, which we used for training the transformers, there is an option to extract the attention map. Since using this option resulted in a bug, we fixed it by changing the source code of the library. Once the issue was fixed, we could save the attention maps.

A.1.14 COMPARING AGAINST OTHER ALIGNMENT PROGRAMS

We compared BetaAlign against the following programs, used with default parameters: MUSCLE v3.8.1551 (Edgar, 2004), MAFFT v7.475 (Katoh & Standley, 2013), PRANK v.150803 (Löytynoja & Goldman, 2008), T-Coffee Version.12.00.7fb08c2 (Notredame et al., 2000), ClustalW 2.1 (Larkin et al., 2007), and DIALIGN dialign2-2 (Morgenstern, 2004).

A.2 SUPPLEMENTARY DISCUSSION

A.2.1 COVERAGE RESULTS

When generating the results of Fig. 4 in the main text, a small fraction of the resulting alignment was invalid, e.g., because of mutations introduced by the transformer to the unaligned sequences. As explained in the New Approach section, changing the tunable parameters of the transformers leads to different deep-learning models. We harnessed this feature of the neural network models and generated an additional transformer for aligning nucleotide sequences as well as an additional transformer for aligning protein sequences (we call each of these transformers "alternative"). When we aligned the invalid cases with the alternative transformers, a large fraction yielded valid alignments. Thus, every time the first "original" transformer fails, the results of the alternative transformer were considered. This methodology substantially increased the coverage, i.e., the percentage of MSAs that were successfully aligned. Specifically, the fraction of datasets that could not be aligned was less than 2% both for nucleotide and protein sequences (Fig. 10).

A.2.2 ADDITIONAL COMMENTS

(1) As each segment translated with the "segmented transformers" is translated independently, different alignments dynamics could be inferred based on this specific segment. This could have an impact on aligning large genomic sequences as different regions in the genome are corresponding to different dynamics as selection powers work differently across the genome.

(2) In this research, we used classical NLP architectures, and loss functions. As we showed in the in Attention Map Comparison section, biological data behave differently from conventional NLP translation tasks, which suggests that there is much room for improvement. Future research should aim to develop NLP tools that better capture the structure of biological macromolecules and their evolution. Specifically, we anticipate the future development of novel architectures of transformers as well as more relevant loss functions.

(3) One of the main challenges of BetaAlign is generalizing the alignments for longer sequences, a well-known hurdle in NLP research. Here, we presented a way to divide the alignments into segments. Further work may explore alternative solutions to this challenge. Different architectures could increase the length of the sentences, and different representations could reduce the number of words in each sentence, e.g., by compressing both the input and output sentences.

A.2.3 ADDITIONAL DATASETS

Table 2 contains the parameters used to generate the different datasets. (a) contains the information on the DNA datasets; (b) contains the information on the protein datasets; (c) contains information about other specific datasets. The random topologies were generated using the program ETE version 3.0. For each dataset, millions of alignments were generated and divided to training data, validation data, and testing data.

A.2.4 DATA FOR SPECIFIC FIGURES

The datasets for Fig. 5 are SND5 and SPD9 for the nucleotide and protein dataset, respectively. To create Fig. 10, we used the same dataset as Fig. 4 in the main text. Of note, we have used the three input sequences of protein (SPD1). In Fig. 6, the transformers that we used were trained on datasets SND1 and SPD1. The test datasets that include spatial variation are SND6 and SPD10 for nucleotide and protein datasets, respectively. Shown in Fig. 7a and Fig. 7c are attention maps of different resolutions extracted from one of the protein alignments composed of seven sequences (dataset PD14).

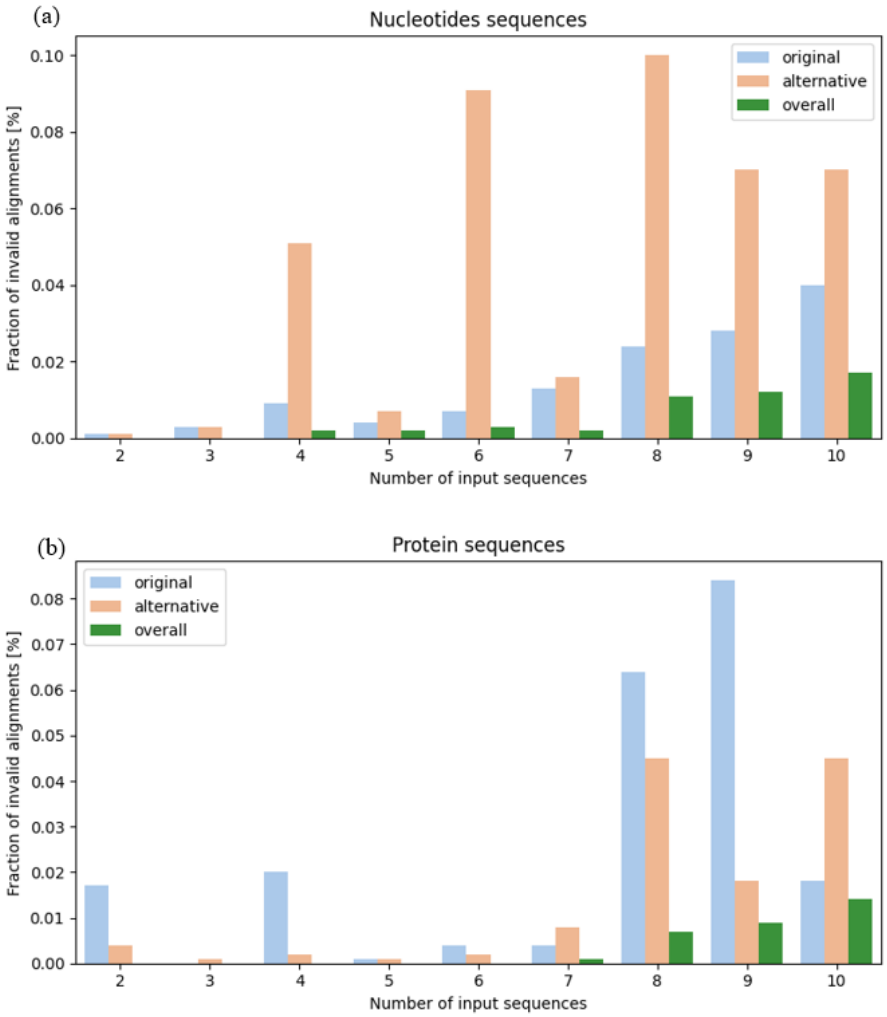

Figure 10: The fraction of invalid alignments for nucleotides (a) and protein (b) sequences as a function of the number of inputs sequences. For each dataset, two different transformers were trained. The fraction of invalid alignments of the "original" and "alternative" transformers are in blue and orange, respectively. In green is the fraction of invalid MSAs shared by both transformers.

Table 2: The table contains the specific information to generate the datasets used in this work. The $R_I\&R_D$ and the $A_I\&A_D$ refer to the indel parameters of the SpartaABC. The root length described in the table is the input of the SpartaABC, which the latter sample uniformly from the min range * 0.8 and the max range * 1.1. The datasets name refers to the order in which the transformers were trained on. For example, the first protein transformer was trained on dataset PD1, then the optimized weights were the starting point of PD2, etc. Similarly, the nucleotides transformer was first trained on ND1. Tables (a), (b) refer to nucleotide and protein datasets, respectively. Table (c) refers to special datasets (hence the "S" at the start of the dataset name). In this table, datasets SND1 and SPD1LG are the datasets used for creating Fig. 4. The other datasets, SND2 to SND5 and SPD2 to SPD9 refers to nucleotide and protein sequences used for the "segmented transformers", respectively. SND6 and SPD10 refers to the misclassification datasets, and thus, the indel parameters are sampled differently for each "domain" (see "Generating Misspecification Dataset" section).

DATASET	BRANCH LENGTH	ROOT LENGTH	$R_I\&R_D$	$A_I\&A_D$	INPUT SEQUENCES NUMBER
ND1	0.03 - 0.1	50 - 60	0.0 - 0.05	1.01 - 2.0	2
ND2	0.03 - 0.3	100 - 300	0.0 - 0.05	1.01 - 2.0	2
ND3	0.3 - 0.6	200 - 300	0.04 - 0.05	1.01 - 2.0	2
ND4	0.1 - 0.3	50 - 60	0.04 - 0.05	1.01 - 2.0	3
ND5	0.15	55	0.5	1.0 - 1.01	3
ND6	0.15	55	0.5	1.01	3
ND7	0.15	55	0.5	1.5	3
ND8	0.05 - 0.1	55	0.0 - 0.05	1.01 – 2	4
ND9	0.05 - 0.1	55	0.03 - 0.05	1.01 – 2	4
ND10	0.07 - 0.1	35 - 45	0.0 - 0.05	1.01 – 2	5
ND11	0.08 - 0.09	37 - 42	0.03 - 0.05	1.01 – 2	5
ND12	0.09	40	0.04	1.3	5
ND13	0.05 - 0.1	55	0.02 - 0.03	1.0 - 1.1	4
ND14	0.9	40	0.01 - 0.02	1.35 - 1.45	5
ND15	0.07 - 0.1	35 - 45	0.0 - 0.05	1.01 – 2	7
ND16	0.07 - 0.1	70 - 80	0.0 - 0.05	1.01 – 2	7

DATASET	BRANCH LENGTH	ROOT LENGTH	$R_I\&R_D$	$A_I\&A_D$	INPUT SEQUENCES NUMBER
PD1	0.03 - 0.05	30 - 40	0.0 - 0.05	1.01 - 2.0	2
PD2	0.05 - 0.1	70 - 80	0.04 - 0.05	1.01 - 2.0	2
PD3	0.1 - 0.3	200 - 250	0.04 - 0.05	1.01 - 2.0	2
PD4	0.03 - 0.1	30 - 40	0.0 - 0.05	1.01 - 2.0	3
PD5	0.1 - 0.2	50 - 60	0.04 - 0.05	1.01 - 2.0	3
PD6	0.15	50	0.05	1.01	3
PD7	0.15	50	0.05	1.5	3
PD8	0.05 - 0.1	30 - 40	0.0 - 0.05	1.01 - 2	4
PD9	0.075	35	0.03	1.07	4
PD10	0.04 - 0.08	30 - 40	0.0 - 0.05	1.01 - 2	5
PD11	0.04 - 0.08	30 - 40	0.0 - 0.05	1.01 - 2	6
PD12	0.1	30 - 40	0.0 - 0.05	1.01 - 2	6
PD13	0.1	30 - 40	0.03 - 0.05	1.01 - 2	6
PD14	0.07 - 0.1	25 – 35	0.0 - 0.05	1.01 - 2	7
PD15	0.08 - 0.09	27 - 32	0.03 - 0.05	1.01 - 2	7
PD16	0.09	30	0.04	1.3	7
PD17	0.05 - 0.1	40	0.0 - 0.05	1.01 - 2	10
PD18	0.07 - 0.1	25 - 35	0.04 - 0.05	1.01 - 2	7

DATASET	TRANSFORMER NAME	ARCHITECTURE	MAX TOKENS	LEARNING RATE	SC-SCORE	COVERAGE
SND1	0.05 - 0.1	40	0.0 - 0.05	1.01 - 2.0	10	
SND2	0.1 - 0.15	600 - 800	0.0 - 0.05	1.01 - 2	3	
SND3	0.1 - 0.15	600 - 800	0.0 - 0.05	1.01 - 2	4	
SND4	0.1 - 0.15	100 - 120	0.0 - 0.05	1.01 - 2	5	
SND5	0.1 - 0.15	600 - 800	0.0 - 0.05	1.01 - 2	5	
SND6	0.05 - 0.1	40	Dynamic	Dynamic	10	
SPD1	0.05 - 0.1	40	0.0 - 0.05	1.01 - 2.0	10	
SPD1LG	0.05 - 0.1	40	0.0 - 0.05	1.01 - 2.0	10	
SPD2	0.08 - 0.15	220 - 280	0.0 - 0.05	1.01 - 2	3	
SPD3	0.08 - 0.15	50 - 120	0.0 - 0.05	1.01 - 2	4	
SPD4	0.08 - 0.15	250 - 350	0.0 - 0.05	1.01 - 2	4	
SPD5	0.08 - 0.2	600 - 800	0.0 - 0.05	1.01 - 2	4	
SPD6	0.08 - 0.2	600 - 800	0.0 - 0.05	1.01 - 2	7	
SPD7	0.08 - 0.2	600 - 800	0.0 - 0.05	1.01 - 2	5	
SPD8	0.1 - 0.15	600 - 800	0.0 - 0.05	1.01 - 2	4	
SPD9	0.1 - 0.15	600 - 800	0.0 - 0.05	1.01 - 2	5	
SPD10	0.05 - 0.1	40	Dynamic	Dynamic	10	
SPD11	0.07 - 0.1	25 - 35	Dynamic	Dynamic	7	