End-to-end learning of multiple sequence alignments with differentiable Smith-Waterman

Samantha Petti [*] Nicholas Bhattacharya [†] Roshan Rao[†] Justas Dauparas [‡]
Neil Thomas[†] Juannan Zhou [§] Alexander M. Rush [¶] Peter K. Koo [‖]
Sergey Ovchinnikov[***]

October 23, 2021

Abstract

Multiple Sequence Alignments (MSAs) of homologous sequences contain information on structural and functional constraints and their evolutionary histories. Despite their importance for many downstream tasks, such as structure prediction, MSA generation is often treated as a separate pre-processing step, without any guidance from the application it will be used for. Here, we implement a smooth and differentiable version of the Smith-Waterman pairwise alignment algorithm that enables jointly learning an MSA and a downstream machine learning system in an end-to-end fashion. To demonstrate its utility, we introduce SMURF (Smooth Markov Unaligned Random Field), a new method that jointly learns an alignment and the parameters of a Markov Random Field for unsupervised contact prediction. We find that SMURF mildly improves contact prediction on a diverse set of protein and RNA families. As a proof of concept, we demonstrate that by connecting our differentiable alignment module to AlphaFold2 and maximizing the predicted confidence metric, we can learn MSAs that improve structure predictions over the initial MSAs. This work highlights the potential of differentiable dynamic programming to improve neural network pipelines that rely on an alignment.

1 Introduction

Multiple Sequence Alignments (MSAs) are commonly used in biology to model evolutionary relationships and the structural/functional constraints within families of proteins and RNA. Alignments are a critical component of the protein structure prediction pipeline [4, 23], help predict the functional effects of mutations [16, 21, 43], and can be used for rational protein design [17]. Creating alignments, however, is a challenging problem. Standard approaches use heuristics for penalizing substitutions and gaps and do not take into account the effects of contextual interactions [41]. For example, these local approaches struggle when aligning large numbers of diverse sequences, and additional measures (such as the introduction of external guide HMMs) must be introduced to obtain reasonable alignments [39]. Finally, each alignment method has a number of hyperparameters

[*]Harvard University
[†]University of California, Berkeley
[‡]University of Washington
[§]University of Florida
[¶]Cornell University
[‖]Cold Spring Harbor Laboratory
[**]Email: so@fas.harvard.edu

which are often chosen on an application-specific basis. This suggests that computational methods that input an MSA could be improved by jointly learning the MSA and training the method.

To understand whether MSA generation can be directly incorporated into learning, rather than treated as a separate preprocessing step, we implemented a smooth and differentiable version of the Smith-Waterman algorithm that can be applied to learn alignments in end-to-end neural network pipelines. The classic version of the Smith-Waterman algorithm outputs a pairwise alignment between two sequences that maximizes the alignment score [40]. We applied the generalized framework for differentiable dynamic programming developed in [29] to build a smooth and differentiable version of Smith-Waterman. We describe the algorithm as "smooth" because it outputs a distribution over alignments rather than a single highest scoring alignment. The smoothness is crucial to (i) make the algorithm differentiable and therefore applicable in end-to-end neural network pipelines, and (ii) allow the method to consider multiple hypothesized alignments simultaneously, which is especially important early in training.

We apply our smooth Smith-Waterman implementation in two neural network pipelines. First, we design an unsupervised contact prediction method that jointly learns an alignment and the parameters of a Markov Random Field (MRF) for RNA and protein. Next we connect our differentiable alignment method to AlphaFold2 to jointly infer an alignment and protein structure [23]. Our main contributions are as follows:

1. We implemented a smooth and differentiable version of Smith-Waterman for local pairwise alignment in JAX [7]. Our implementation includes options for an affine gap penalty, a temperature parameter that controls the relaxation from the highest scoring path (i.e. smoothness), and both global and local alignment settings. Our code is freely available and can be applied in any end-to-end neural network pipeline written in JAX or tensorflow. Moreover, we give a self-contained description of our implementation and its mathematical underpinnings, providing a template for future implementations in other languages.

2. We designed a contact prediction method called *Smooth Markov Unaligned Random Field* (SMURF) that inputs unaligned sequences and jointly learns an alignment and MRF parameters. Our pipeline begins with a *learned alignment module* (LAM). For each sequence, a convolutional architecture produces a matrix of match scores between the sequence and a reference. Unlike a subsitution matrix typically input to Smith-Waterman, these scores account for the local k-mer context of each residue. Next we apply smooth Smith-Waterman to these similarity matrices to align each sequence to the reference (Figure 2). To train SMURF on a batch of sequences, we apply LAM to a produce an expected alignment, which we use to form an MSA for the batch. Then, we compute masked language modeling loss under the MRF for this MSA and update parameters (Figure 10). We show that SMURF outperforms a neural version of GREMLIN for protein and RNA contact prediction on a diverse set of families.

3. To demonstrate the utility of a differentiable alignment layer, we modify AlphaFold2 [23], replacing the MSA with the LAM. For a given set of unaligned related sequences, we backprop through AlphaFold to update the parameters of LAM, maximizing the predicted LDDT. Doing so improves the structure prediction over our initial input MSA for 4 out of 5 targets.

1.1 Related work

Differentiable Dynamic Programming in NLP. Differentiable dynamic programming algorithms are needed in order to model combinatorial structures in a way that allows backpropagation of gradients. Such algorithms have been used in NLP to build neural models for parsing [13], grammar induction [27], speech [8], and more. Smooth relaxations of argmax and other non-differentiable functions can enable differentiation through dynamic programs. More generally, Mensch and Blondel leverage semirings to provide a unified framework for constructing differentiable operators from a general class of dynamic programming algorithms [29]. This work has been incorporated into the Torch-Struct library [37] to enable composition of automatic differentiation and neural network primitives and was also recently implemented in Julia [42]. We base our smooth Smith-Waterman implementation off of this work.

Smooth and differentiable alignment in computational biology Before end-to-end learning was common, computational biologists used pair HMMs to express probability distributions over pairwise alignments [12]. The forward algorithm applied to a pair HMM can be viewed as a smoothed version of Smith-Waterman. Later, a differentiable kernel-based method for alignment was introduced [38]. More recently, Morton et al. implemented a differentiable version of the Needleman-Wunsch algorithm for global pairwise alignment [34, 31]. Our implementation has several advantages: (i) vectorization makes our code faster (Appendices B.1 and B.4)), (ii) we implemented local alignment and an affine gap penalty (Appendix B.5), and (iii) due to the way gaps are parameterized, the output of [31] can not be interpreted as an expected alignment (Appendix B.3).

Language models, MRFs, and alignments. Previous work combining language model losses with alignment of biological sequences place the alignment layer at the end of the pipeline. Bepler et al. first pretrain a bidirectional RNN language model, then freeze this model and train a downstream model using a pseudo-alignment loss [5]. Similarly, Morton et al. use a pretrained language model to parametrize the the alignment scoring function [31]. Their loss, however, is purely supervised based on ground-truth structural alignments. For RNA, a transformer embedding has been trained jointly with a masked language modeling and structural alignment [1]. In contrast to all of these papers, our alignment layer is in the middle of the pipeline and is trained end-to-end.

Joint modeling of alignments and Potts models has been explored. Kinjo includes insertions and deletions into a Potts model using techniques from statistical physics [28]. With the goal of designing generative classifiers for protein homology search, two other works jointly infer HMM and Potts parameters through importance sampling [44] and message passing [32].

2 Smooth Smith-Waterman

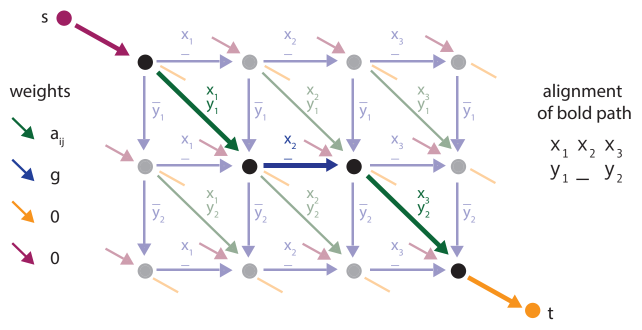

Figure 1: The alignment graph for sequences $X = x_1x_2x_3$ and $Y = y_1y_2$. Edge labels describe the corresponding aligned pair, and colors indicate the weights. All red edges start at the source s, and all orange edges end at the sink t. The bold path corresponds to the alignment of X and Y written on the right.

Pairwise sequence alignment can be formulated as the task of finding the highest scoring path through a directed graph in which edges correspond to an alignment of two particular residues or to a gap. The edge weights are match scores for the corresponding residues or the gap penalty, and the score of the path is the sum of the edge weights. The Smith-Waterman algorithm is a dynamic programming algorithm that returns a path with the maximal score. Smooth Smith-Waterman instead finds a probability distribution over paths in which higher scoring paths are more likely.

Figure 1 illustrates an alignment graph. For sequences $x_1, x_2, \dots x_{\ell_x}$ and $y_1, y_2, \dots, y_{\ell_y}$, the vertex set contains grid vertices v_{ij} for $0 \leq i \leq \ell_x$ and $0 \leq j \leq \ell_y$, a source s, and a sink t. The directed edges are defined so that each path from s to t corresponds to a local alignment of the sequences. The table below describes the definitions, meanings, and weights of the edges.

Edge	Meaning	Weight
$v_{i-1,j-1} \to v_{i,j}$	x_i and y_j are aligned	x_i,y_j alignment score a_{ij}
$v_{i,j-1} \to v_{i,j}$	y_j is aligned with the gap character _	gap penalty g
$v_{i-1,j} \to v_{i,j}$	x_i is aligned with the gap character _	gap penalty g
$s \to v_{i,j}$	x_k for $k \leq i$ and y_k for $k \leq j$ are excluded	0
$v_{i,j} \to t$	x_k for $k > i$ and y_k for $k > j$ are excluded	0

The Smith-Waterman algorithm iteratively computes the highest score of a path ending at each vertex and returns the highest scoring path ending at t. Let $w(u \to v)$ denote the weight of the edge $u \to v$, and let $N^-(v) = \{u \,|\, u \to v \text{ is an edge}\}$ denote the incoming neighbors of v. Let $f(v)$ be the value of the highest scoring path from s to v. Taking $f(s) = 0$, we compute

$$f(v) = \max_{u \in N^-(v)} \{f(u) + w(u \to v)\}, \text{ so } f(v_{i,j}) = \max\{f(v_{i-1,j-1}) + a_{ij}, f(v_{i,j-1}) + g, f(v_{i-1,j}) + g\}.$$

A path with the highest score is computed by starting at the sink t and tracing backward along the edges that achieve the maxima. (For further explanation see Chapter 2 of [12] or [40]).

Following the general differentiable dynamic programming framework introduced in [29], we implement a smoothed version of Smith-Waterman. We compute a smoothed version of the function f, which we denote f^S, by replacing the `max` with `logsumexp`. We again take $f^S(s) = 0$, and define

$$f^S(v) = \log\left(\sum_{u \in N^-(v)} \exp\left(f^S(u) + w(u \to v)\right)\right). \tag{1}$$

We use these smoothed scores and the edge weights to define a probability distribution over paths in G, or equivalently local alignments.

Definition 1. *Given an alignment graph $G = (E, V)$, define a random walk starting at vertex t that traverses edges of G in reverse direction according to transitions probabilities*

$$\mathbb{T}(v \to u) = \frac{\exp\left(f^S(u) + w(u \to v)\right)}{\sum_{u' \in N^-(v)} \exp\left(f^S(u') + w(u' \to v)\right)}$$

and ends at the absorbing vertex s. Let μ_G be the probability distribution over local alignments in which the probability of an alignment A is equal to the probability that the random walk follows the reverse of the path in G corresponding to A.

Under the distribution μ_G, the probability that residues x_i and y_j are aligned can be formulated as a derivative. Mensch and Blondel describe this relationship in generality for differentiable dynamic programming on directed acyclic graphs [29]. We state their result as it pertains to our context and provide a proof in our notation in Appendix B.2.

Proposition 1 (Proposition 3 of [29]). *Let G be an alignment graph and μ_G be the corresponding probability distribution over alignments. Then*

$$\mathbb{P}_{\mu_G}(\ x_i \textit{ and } y_j \textit{ are aligned }) = \frac{\partial f^S(t)}{\partial w(v_{i-1,j-1} \to v_{i,j})} = \frac{\partial f^S(t)}{\partial a_{ij}}.$$

Our implementation of Smooth Smith-Waterman. Our implementation of the Smooth Smith-Waterman algorithm takes as input a gap penalty g and a matrix whose values a_{ij} correspond to the alignment score for residues x_i and y_j. We use JAX due to its JIT ('just in time') compilation and automatic differentiation features [7]. Our implementation leverages vectorization to iteratively compute $f^S(t)$ (e.g. the values $f^S(v_{ij})$ for each fixed value of $i + j$ are computed in tandem, see [15] and Appendix B.4). Automatic differentation in JAX then computes the derivative $f^S(t)$ with respect to the input parameters a_{ij}. Since our implementation is fully differentiable, it can be composed with JAX or tensorflow implementations of neural networks in end-to-end pipelines.

We introduce four other features, detailed in Appendix B.5. A *temperature* parameter governs the extent to which μ_G is concentrated on the highest scoring paths. In the *affine gap* mode, the first gap in a streak incurs an "open" gap penalty and all subsequent gaps incur an "extend" gap penalty. A *restrict turns* option corrects for the algorithm's inherent bias towards alignments near the diagonal. Finally, we implement Needleman-Wunsch global alignment. In Appendix B.1, we demonstrate that our implementation is faster than the Needleman-Wunsch implementation of [31].

3 Application to contact prediction

GREMLIN is a probabilistic model of protein variation that uses the MSA of a protein family to estimate parameters of a Markov Random Field (MRF) of the form

$$\mathbb{P}(X = x) = \frac{1}{Z}\exp\left(E\left(x; v, w\right)\right) \quad \text{where} \quad E\left(x; v, w\right) = \sum_{i=1}^{\ell}\left[v_i(x_i) + \sum_{j=1}^{\ell} w_{ij}(x_i, x_j)\right] \tag{2}$$

where ℓ is the number of columns in the MSA, v_i represents the amino acid propensities for position i, w_{ij} is the pairwise interaction matrix for positions i and j, and Z is the partition function (the value $E(\cdot\,; v, w)$ summed over all sequences x). Typically the model is trained by maximizing the pseudolikihood of observing all sequences in the alignment [25, 35, 14, 4]. Here we follow the approach of [6, 36] and use *Masked Language Modeling* (MLM) to find the parameters w and b. The pairwise terms w_{ij} can be used to predict contacts by reducing each matrix w_{ij} into a single value that indicates the extent to which positions i and j are coupled.

Since GREMLIN relies on an input MSA, one would expect that improved alignments would yield better contact prediction results. To test this idea, we designed a pipeline for training a GREMLIN-like model that inputs unaligned sequences and jointly learns the MSA and MRF parameters. We call our method **S**mooth **M**arkov **U**naligned **R**andom **F**ield or SMURF.

3.1 Learned alignment module

The key to improving a Smith-Waterman alignment is finding the right input matrix of alignment scores $a = (a_{ij})_{i \le \ell_x, j \le \ell_y}$. Typically, when Smith-Waterman is used for pairwise alignment the alignment score between positions i and j, a_{ij}, is given by a BLOSUM or PAM score for the pair of residues X_i and Y_j [2, 10, 19]. This score reflects how likely it is for one amino acid to be substituted for another, but does not acknowledge the context of each residue in the sequence.

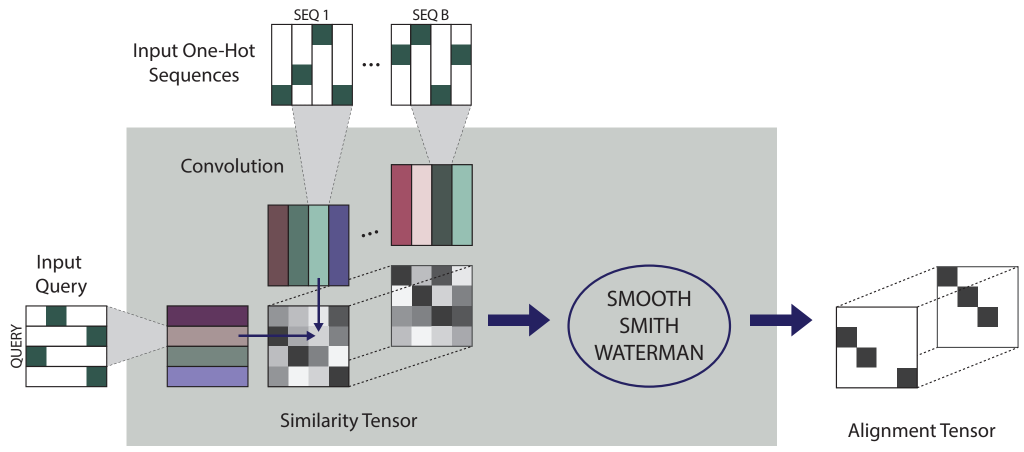

Figure 2: Learned alignment module (LAM). Residues are encoded as vectors through a convolution of the one-hot representations of the sequences. A similarity tensor is constructed by taking the dot product of the vectors for each sequence with the vectors for the query sequence. Smooth Smith-Waterman is applied, yielding smooth pairwise alignments between each sequence and the query sequence.

We introduce the learned alignment module (LAM) that adaptively learns a context-dependent alignment score matrix a_{ij}. The value a_{ij} expresses the similarity between X_i in the context of $X_{i-w}, \dots X_i, \dots X_{i+1}$ and Y_j in the context of $Y_{j-w}, \dots Y_j, \dots Y_{j+w}$. We represent position i in sequence X as a vector v_i^X obtained by applying a convolutional layer of window size w to a one-hot encoding of X_i and its neighbors. The value a_{ij} in our input matrix is the dot product of the corresponding vectors, $a_{ij} = v_i^X \cdot v_j^Y$. Figure 2 illustrates a batched version of LAM. Our use of smooth Smith-Waterman makes this process entirely differentiable, enabling us to plug our alignment into a downstream module, compute a loss function, and train the pipeline end-to-end.

3.2 SMURF protein and RNA contact prediction

SMURF takes as input unaligned sequences in a family and learns both (i) the LAM convolutions and (ii) the parameters of the MRF that are used to predict contacts. SMURF has two phases, each beginning with the LAM. Initial convolutions are learned in BasicAlign. Then in TrainMRF, a MLM objective is used to learn MRF parameters and update the convolutions (Figure 10). We compare SMURF to GREMLIN trained with masked language modeling (MLM-GREMLIN) [6]. The architecture of MLM-GREMLIN is the similar to TrainMRF step of SMURF, except that an alignment is part of the input instead of computed via the LAM.

We trained and evaluated our model on a diverse set of protein families, as described in Appendices A.1 and C.3. To evaluate the accuracy, we computed the area under the curve (AUC) for a plot of t versus the fraction of the top t predicted contacts that are correct. Figure 3a illustrates that SMURF mildly outperforms MLM-GREMLIN with a median AUC improvement of 0.007. In Appendix C.2, we show a comparable result for families with at most 128 sequences (Figure 13), and we illustrate some contact maps and alignments produced by SMURF (Figures 11 and 12).

Next, we applied SMURF to 17 non-coding RNA families from Rfam [24] that had a corresponding structure in PDB (Appendix A.2). Overall, we observe that SMURF slightly outperforms MLM-GREMLIN with a median AUC improvement of 0.02 (Figure 3b). We hypothesize that SMURF generates fewer false positive predictions in seemingly random locations because the LAM finds better alignments. The contact predictions in Figure 3b are further discussed in Appendix C.2.

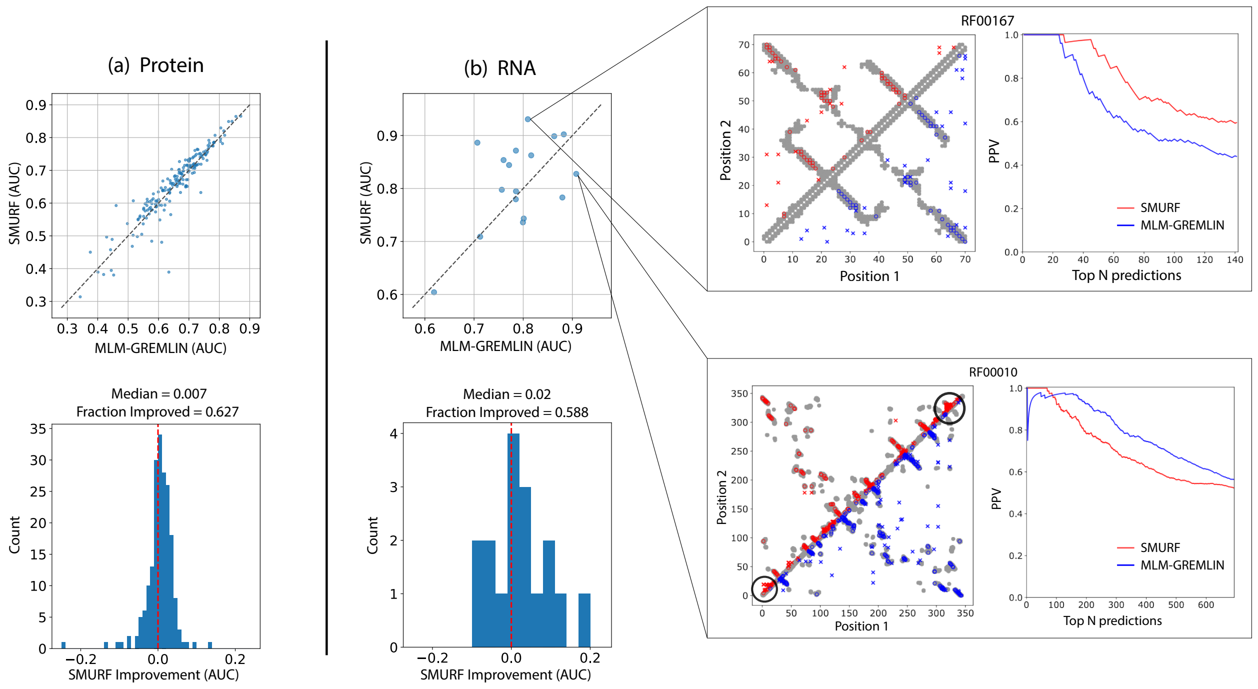

Figure 3: Performance comparison on (a) protein and (b) non-coding RNA. (Top) Scatter plots of the AUC of the top L predicted contacts for SMURF versus MLM-GREMLIN. (Bottom) Histograms of the difference in AUC between SMURF and MLM-GREMLIN. (Right) Comparison of contact predictions and the positive predictive value (PPV) for different numbers of top N predicted contacts, with N ranging from 0 to $2L$, for SMURF (red) and MLM-GREMLIN (blue) for Rfam family RF00010 (Ribonuclease P.) and RF00167 (Purine riboswitch). Gray dots represent PDB-derived contacts, circles represent a true positive prediction, and x represents a false positive prediction. For contact predictions for RFAM00010, the black circles highlight a concentration of false positive predictions.

4 Backprop through AlphaFold2

As a proof of concept, we selected five CASP14 domains where the structure prediction quality from AlphaFold was especially sensitive to how the MSA was constructed. We reasoned that the quality was poor due to issues in the MSA and by realigning the sequences using AlphaFold's confidence metrics we may be able to improve on the prediction quality.

For each of the five selected CASP targets, separate LAM parameters were fit to maximize the predicted confidence metric. (See Appendix A.3.) This includes the predicted LDDT (pLDDT) and alignment error as returned by AlphaFold's "model_3_ptm". For all five targets, we were able to improve on the pLDDT. However only 4 of the 5 targets showed an improvement in RMSD.

5 Discussion

In this work we explored the composition of alignment in a pipeline that can be trained end-to-end without usage of any existing alignment software or ground-truth alignments. This highlights the feasibility of pipelines which compose core bioinformatics workflows with downstream machine learning systems and learn jointly over all components. Our smooth Smith-Waterman implementation is designed to be usable and efficient, and we hope it will enable experimentation with alignment modules in other applications of machine learning to biological sequences.

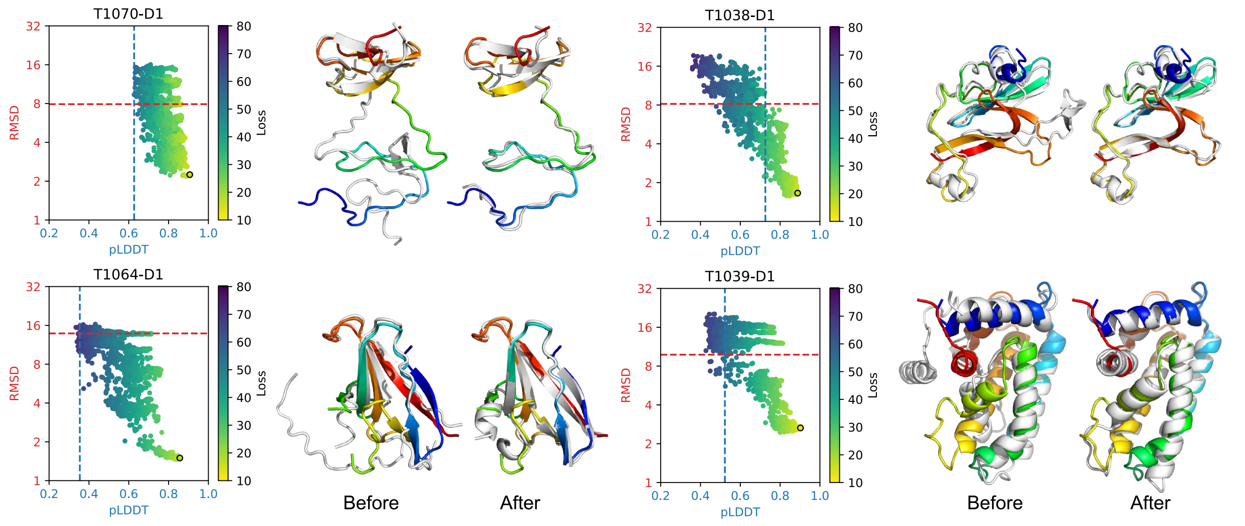

Figure 4: Learnt MSA with higher AlphaFold confidence results in improved structure prediction. The scatter plot shows the sampled pLDDT and RMSD across all trajectories. For example trajectories see Figure 15. The dotted lines show the initial pLDDT and RMSD (root-mean-squared-distance to native structure) using the MSA from MMseqs2. For each CASP14 target, the native structure is rainbow colored, and the predictions are overlaid and colored white. After maximizing the confidence metric, the structure with the max pLDDT (circled) is shown in the after column. For target T1043-D1 that failed to improve see Figure 16.

There is ample opportunity for future work to systematically compare architectures for the scoring function in smooth Smith-Waterman. The use of convolutions led to relatively simple training dynamics, but other inductive biases induced by recurrent networks, attention mechanisms, or hand-crafted architectures could capture other signal important for alignment scoring. We also hope that the use of these more powerful scoring functions enables applications in remote homology search, structure prediction, or studies of protein evolution.

Besides MSAs, there are numerous other discrete structures essential to analysis of biological sequences. These include Probabilistic Context Free Grammars used to model RNA Secondary Structure [33] and Phylogenetic Trees used to model evolution. Designing differentiable layers that model meaningful combinatorial latent structure in evolution and biophysics is an exciting avenue for further work in ML and biology. For example, structured attention [26] could be fruitful to explore given the recent success of attention-based models for protein structure prediction.

Acknowledgements. We thank Sean Eddy for pointing out the need for a restrict turns feature and Jake VanderPlas for supplying JAX code that efficiently rotates a matrix (as in Figure 7b). Computational analyses were performed with assistance from the US National Institutes of Health Grant S10OD028632-01 and the Cannon cluster supported by the FAS Division of Science, Research Computing Group at Harvard University. SP was supported by the NSF-Simons Center for Mathematical and Statistical Analysis of Biology at Harvard (award #1764269). NB was supported in part by NIH grant R35-GM134922 and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. PKK was supported in part by the Simons Center for Quantitative Biology at Cold Spring Harbor Laboratory and the Developmental Funds from the Cancer Center Support Grant 5P30CA045508. SO is supported by NIH Grant DP5OD026389, NSF Grant MCB2032259 and the Moore–Simons Project on the Origin of the Eukaryotic Cell, Simons Foundation 735929LPI, https://doi.org/10.46714/735929LPI.

References

[1] Manato Akiyama and Yasubumi Sakakibara. Informative rna-base embedding for functional rna structural alignment and clustering by deep representation learning. *bioRxiv*, 2021.

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

[3] Ivan Anishchenko, Sergey Ovchinnikov, Hetunandan Kamisetty, and David Baker. Origins of coevolution between residues distant in protein 3d structures. *Proceedings of the National Academy of Sciences*, 114(34):9122–9127, 2017.

[4] Sivaraman Balakrishnan, Hetunandan Kamisetty, Jaime G Carbonell, Su-In Lee, and Christopher James Langmead. Learning generative models for protein fold families. *Proteins: Structure, Function, and Bioinformatics*, 79(4):1061–1078, 2011.

[5] Tristan Bepler and Bonnie Berger. Learning protein sequence embeddings using information from structure. In *International Conference on Learning Representations*, 2018.

[6] Nicholas Bhattacharya, Neil Thomas, Roshan Rao, Justas Daupras, Peter Koo, David Baker, Yun S Song, and Sergey Ovchinnikov. Single layers of attention suffice to predict protein contacts. *bioRxiv*, 2020.

[7] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[8] Xingyu Cai and Tingyang Xu. Dtwnet: a dynamic timewarping network. *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 32, 2019.

[9] Justas Dauparas, Haobo Wang, Avi Swartz, Peter Koo, Mor Nitzan, and Sergey Ovchinnikov. Unified framework for modeling multivariate distributions in biological sequences. *arXiv preprint arXiv:1906.02598*, 2019.

[10] Margaret O Dayhoff and Richard V Eck. *Atlas of protein sequence and structure.* National Biomedical Research Foundation., 1972.

[11] Stanley D Dunn, Lindi M Wahl, and Gregory B Gloor. Mutual information without the influence of phylogeny or entropy dramatically improves residue contact prediction. *Bioinformatics*, 24(3):333–340, 2008.

[12] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[13] Greg Durrett and Dan Klein. Neural crf parsing. *arXiv preprint arXiv:1507.03641*, 2015.

[14] Magnus Ekeberg, Cecilia Lövkvist, Yueheng Lan, Martin Weigt, and Erik Aurell. Improved contact prediction in proteins: Using pseudolikelihoods to infer Potts models. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 87(1), 1 2013.

[15] Michael Farrar. Striped smith–waterman speeds database searches six times over other simd implementations. *Bioinformatics*, 23(2):156–161, 2007.

[16] Matteo Figliuzzi, Hervé Jacquier, Alexander Schug, Oliver Tenaillon, and Martin Weigt. Co-evolutionary landscape inference and the context-dependence of mutations in beta-lactamase tem-1. *Molecular biology and evolution*, 33(1):268–280, 2016.

[17] Adi Goldenzweig, Moshe Goldsmith, Shannon E Hill, Or Gertman, Paola Laurino, Yacov Ashani, Orly Dym, Tamar Unger, Shira Albeck, Jaime Prilusky, et al. Automated structure- and sequence-based design of proteins for high bacterial expression and stability. *Molecular cell*, 63(2):337–346, 2016.

[18] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3):705–708, 1982.

[19] Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[20] Andrea Hildebrand, Michael Remmert, Andreas Biegert, and Johannes Söding. Fast and accurate automatic structure prediction with hhpred. *Proteins: Structure, Function, and Bioinformatics*, 77(S9):128–132, 2009.

[21] Thomas A Hopf, John B Ingraham, Frank J Poelwijk, Charlotta PI Schärfe, Michael Springer, Chris Sander, and Debora S Marks. Mutation effects predicted from sequence co-variation. *Nature biotechnology*, 35(2):128–135, 2017.

[22] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Applying and improving alphafold at casp14. *Proteins*, 2021.

[23] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[24] Ioanna Kalvari, Eric P Nawrocki, Nancy Ontiveros-Palacios, Joanna Argasinska, Kevin Lamkiewicz, Manja Marz, Sam Griffiths-Jones, Claire Toffano-Nioche, Daniel Gautheret, Zasha Weinberg, et al. Rfam 14: expanded coverage of metagenomic, viral and microrna families. *Nucleic Acids Research*, 49(D1):D192–D200, 2021.

[25] Hetunandan Kamisetty, Sergey Ovchinnikov, and David Baker. Assessing the utility of coevolution-based residue–residue contact predictions in a sequence-and structure-rich era. *Proceedings of the National Academy of Sciences*, 110(39):15674–15679, 2013.

[26] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured attention networks. *arXiv preprint arXiv:1702.00887*, 2017.

[27] Yoon Kim, Chris Dyer, and Alexander M Rush. Compound probabilistic context-free grammars for grammar induction. *arXiv preprint arXiv:1906.10225*, 2019.

[28] Akira R Kinjo. A unified statistical model of protein multiple sequence alignment integrating direct coupling and insertions. *Biophysics and physicobiology*, 13:45–62, 2016.

[29] Arthur Mensch and Mathieu Blondel. Differentiable dynamic programming for structured prediction and attention. In *International Conference on Machine Learning*, pages 3462–3471. PMLR, 2018.

[30] Milot Mirdita, Sergey Ovchinnikov, and Martin Steinegger. Colabfold-making protein folding accessible to all. *bioRxiv*, 2021.

[31] Jamie Morton, Charlie Strauss, Robert Blackwell, Daniel Berenberg, Vladimir Gligorijevic, and Richard Bonneau. Protein structural alignments from sequence. *BioRxiv*, 2020.

[32] Anna Paola Muntoni, Andrea Pagnani, Martin Weigt, and Francesco Zamponi. Aligning biological sequences by exploiting residue conservation and coevolution. *Physical Review E*, 102(6):062409, 2020.

[33] Eric P Nawrocki and Sean R Eddy. Infernal 1.1: 100-fold faster rna homology searches. *Bioinformatics*, 29(22):2933–2935, 2013.

[34] Saul B Needleman and Christian 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, 1970.

[35] Sergey Ovchinnikov, Hetunandan Kamisetty, and David Baker. Robust and accurate prediction of residue–residue interactions across protein interfaces using evolutionary information. *elife*, 3:e02030, 2014.

[36] Roshan Rao, Joshua Meier, Tom Sercu, Sergey Ovchinnikov, and Alexander Rives. Transformer protein language models are unsupervised structure learners. In *International Conference on Learning Representations*, 2020.

[37] Alexander M Rush. Torch-struct: Deep structured prediction library. *arXiv preprint arXiv:2002.00876*, 2020.

[38] Hiroto Saigo, Jean-Philippe Vert, and Tatsuya Akutsu. Optimizing amino acid substitution matrices with a local alignment kernel. *BMC bioinformatics*, 7(1):1–12, 2006.

[39] Fabian Sievers and Desmond G Higgins. Clustal omega. *Current protocols in bioinformatics*, 48(1):3–13, 2014.

[40] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[41] Martin Steinegger, Markus Meier, Milot Mirdita, Harald Vöhringer, Stephan J Haunsberger, and Johannes Söding. Hh-suite3 for fast remote homology detection and deep protein annotation. *BMC bioinformatics*, 20(1):1–15, 2019.

[42] Michael Stock. Learning to align with differentiable dynamic programming. `https://www.youtube.com/watch?v=6a07Z6Plp_k`, 2021.

[43] Laksshman Sundaram, Hong Gao, Samskruthi Reddy Padigepati, Jeremy F McRae, Yanjun Li, Jack A Kosmicki, Nondas Fritzilas, Jörg Hakenberg, Anindita Dutta, John Shon, et al. Predicting the clinical impact of human mutation with deep neural networks. *Nature genetics*, 50(8):1161–1170, 2018.

[44] Grey W Wilburn and Sean R Eddy. Remote homology search with hidden potts models. *PLOS Computational Biology*, 16(11):e1008085, 2020.

A Data and code availability

Our code and a detailed description of the data we used is available at: `https://github.com/spetti/SMURF`.

A.1 Protein analysis

For our analysis on proteins, we used the MSAs and contact maps collected in [3]. For training and initial tests, we used a reduced redundancy subset of 383 families constructed in [9]. Each family has least 1K effective sequences, and there is no pair of families with an E-value greater that 1e-10, as computed by an HMM-HMM alignment [20]. A random 190 families were used as the training set to indentify quality hyperparameters of the model. The remaining 193 families served as the test set and are represented in Figure 3a, with the exceptions of two outlier families 4X9JA (SMURF AUC = 0.0748, MLM-GREMLIN AUC = 0.0523) and 2YN5A (SMURF AUC = 0.135, MLM-GREMLIN AUC = 0.145). Figure 13 includes data from 99 families from [20] that have at most 128 sequences. A list of the families used in each setting is available in our GitHub repository.

A.2 Non-coding RNA analysis

For each non-coding RNA, we aligned the RNA sequence in the PDB along with the corresponding Rfam sequences to an appropriate Rfam covariance model using Infernal [33]. We then analyzed these sequences using the same procedure outlined for proteins. We evaluated the efficacy of the predicted contact maps using the PDB-derived contact map, where two nucleotides are classified as in contact if the minimum atomic distance is below 8 angstrom. A list of the families used is available in our GitHub repository.

A.3 AlphaFold experiment

For our case study, the initial multiple sequence alignments (MSA) were obtained from MMseqs2 webserver as implemented in ColabFold [30]. After trimming the MSAs to their official domain definition, they were further filtered to reduce redundancy to 90 percent and to remove sequences that do not cover at least 75 percent of the domain length, using HHfilter [41]. Continuous domains under 200 in length, with at least 20 sequences, RMSD (root-mean-squared-distance) greater than 5 angstroms and the predicted LDDT (confidence metric) below 75, were selected for the experiment. We include one discontinuous targets T1064-D1 (SARS-CoV-2 ORF8 accessory protein) with only 16 sequences as an extra case study, as this was a particularly difficult CASP target that required manual MSA intervention, guided by pLDDT, to predict well [22]. The filtered MSAs were unaligned (gaps removed, deletions relative to query added back in) and padded to the max length.

When the number of sequences is low, we find the optimization to be especially sensitive to parameter initialization. To increase robustness, 30 independent optimization trajectories with 100 iterations each were carried out using ADAM. For each trajectory, a different seed and learning rate were used. The learning rates include 1E-2, 1E-3 and 1E-4. See Figure 15 and Figure 16 for all the trajectories.

B Smooth Smith-Waterman

B.1 Speed test

Our speed benchmark indicates that our implementations are faster than the smooth Needleman-Wunsch implementation in [31]. Moreover, comparison between a vectorized and naive version of our code shows that vectorization substantially reduces the runtime. We compare the time for a forward pass as well as for both the forward and backward passes. The latter is relevant when using the method in a neural network pipeline requiring backprogation.

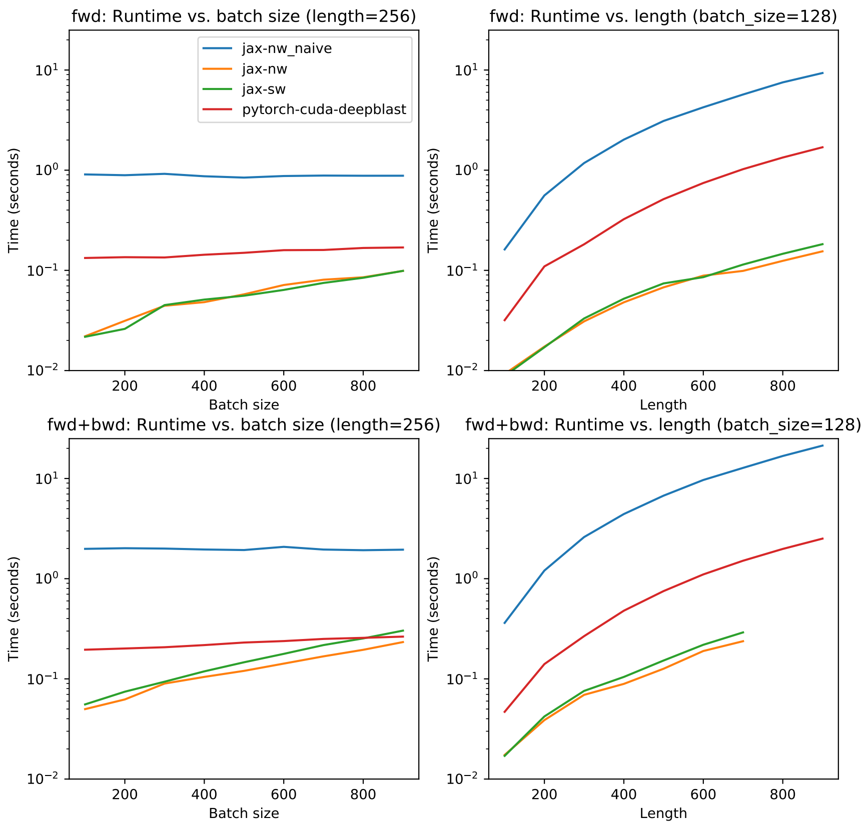

Figure 5: Runtime comparisons between the Needleman-Wunsch implementation in [31] our JAX implementations of smooth Smith-Waterman (green), smooth Needleman-Wunsch (orange) and a naive non-vectorized Needleman-Wunsch (blue). Top plots report time for a forward pass, and the bottom plots report time for a forward and backward pass.

B.2 Proof of Proposition 1 (probabilistic interpretation of the gradient)

For completeness, we now repeat the proof of Proposition 1 given in [29] for the special case of Smooth Smith-Waterman. Proposition 1 gives a probabilistic interpretation of the gradient $f^S(t)$ with respect to the edge weights a_{ij}. We first give a probabilistic interpretation of the gradient $f^S(t)$ with respect to the vertex scores $f^S(v_{ij})$.

Proposition 2. *Let G be an alignment graph. With respect to the random walk described in Definition 1,*

$$\mathbb{P}(\ v \textit{ is visited }) = \frac{\partial f^S(t)}{\partial f^S(v)}.$$

Proof. Let $N^+(v) = \{u \,|\, v \to u \text{ is an edge in } G\}$ denote the outgoing neighborhood of v. Let $u_1, \ldots u_n$ denote the vertices of G in a reverse topological order. We prove the statement by induction with respect to this order. Note $u_1 = t$, and $\mathbb{P}(\ t \text{ is visited }) = \frac{\partial f^S(t)}{\partial f^S(t)} = 1$. Assume that for all $1 \le i \le j$, $\mathbb{P}(\ u_i \text{ is visited }) = \frac{\partial f^S(t)}{\partial f^S(u_i)}$. Observe

$$\begin{aligned}
\frac{\partial f^S(t)}{\partial f^S(u_{j+1})} &= \sum_{u' \in N^+(u_{j+1})} \frac{\partial f^S(t)}{\partial f^S(u')} \frac{\partial f^S(u')}{\partial f^S(u_{j+1})} \\
&= \sum_{u' \in N^+(u_{j+1})} \mathbb{P}(\ u' \text{ is visited }) \frac{\partial}{\partial f^S(u_{j+1})} \log \left(\sum_{u'' \in N^-(u')} \mathsf{exp}\left(f^S(u'') + w(u'' \to u')\right) \right) \\
&= \sum_{u' \in N^+(u_{j+1})} \mathbb{P}(\ u' \text{ is visited }) \frac{\mathsf{exp}\left(f^S(u_{j+1}) + w(u_{j+1} \to u')\right)}{\sum_{u'' \in N^-(u')} \mathsf{exp}\left(f^S(u'') + w(u'' \to u')\right)} \\
&= \sum_{u' \in N^+(u_{j+1})} \mathbb{P}(\ u' \text{ is visited }) \mathbb{T}\left(u' \to u_{j+1}\right) \\
&= \mathbb{P}(\ u_{j+1} \text{ is visited }),
\end{aligned}$$

where in the second equality we apply the inductive hypothesis. □

Proof of Proposition 1. It suffices to show that for each directed edge $u \to v$ in G

$$\frac{\partial f^S(t)}{\partial w(u \to v)} = \mathbb{P}(\ \text{edge } u \to v \text{ is traversed }\ )$$

where the traversal occurs from v to u in the random walk. Observe

$$\begin{aligned}
\frac{\partial f^S(t)}{\partial w(u \to v)} &= \frac{\partial f^S(t)}{\partial f^S(v)} \frac{\partial f^S(v)}{\partial w(u \to v)} \\
&= \mathbb{P}(\ v \text{ is visited }) \frac{\partial}{\partial w(u \to v)} \log \left(\sum_{u' \in N^-(v)} \mathsf{exp}\left(f^S(u') + w(u' \to v)\right) \right) \\
&= \mathbb{P}(\ v \text{ is visited }) \frac{\mathsf{exp}\left(f^S(u) + w(u \to v)\right)}{\sum_{u' \in N^-(v)} \mathsf{exp}\left(f^S(u') + w(u' \to v)\right)} \\
&= \mathbb{P}(\ v \text{ is visited }) \mathbb{T}\left(v \to u\right) \\
&= \mathbb{P}(\ \text{edge } u \to v \text{ is traversed }).
\end{aligned}$$

□

B.3 Difference in Needleman-Wunsch implementation of [31]

The authors of [31] implement a differentiable version of the Needleman-Wunsch global alignment algorithm [34]. Their implementation differs from ours in how gaps are parameterized. Consequently, their output indicates where gaps or matches are likely, whereas our output expresses matches in an expected alignment.

The authors of [31] define

$$v_{i,j} = \mu_{i,j} + \max_{\Omega}\left(v_{i-1,j-1}, g_{i,j} + v_{i-1,j}, g_{i,j} + v_{i,j-1}\right),$$

where $g_{i,j}$ is the gap penalty for an insertion or deletion at i or j, $\mu_{i,j}$ is the alignment score for X_i and Y_j, and $\max_{\Omega}(x) = \log\left(\sum_i \exp\left(x_i\right)\right)$ (see Appendix A of [31]). The values $v_{i,j}$ are analogous to our definition f^S on grid vertices (Equation (1)) with match scores $\mu_{i,j} = a_{i,j}$,

$$f^S(v_{i,j}) = \max_{\Omega}\left(f^S(v_{i-1,j-1}) + \mu_{i,j}, f^S(v_{i,j-1}) + g, f^S(v_{i-1,j}) + g\right).$$

In the alignment graph for their formulation, gap edges have weight $\mu_{i,j} + g_{i,j}$. In our alignment graph, gap edges have weight g; the match score $\mu_{i,j}$ does not play a role, and our gap penalty is not position dependent.

Their code outputs the derivatives $\frac{\partial v_{N,M}}{\partial \mu_{i,j}}$. The derivative $\frac{\partial v_{N,M}}{\partial \mu_{i,j}}$ is high whenever the dominant alignment path uses an edge whose weight includes $\mu_{i,j}$; this includes the edges that corresponds to gaps. In contrast, in our formulation $a_{i,j} = \mu_{i,j}$ appears on the match edge only, and so $\frac{\partial f^S(t)}{\partial a_{i,j}}$ is high only when the dominant alignment path uses the edge corresponding to a match. Proposition 1 establishes that $\frac{\partial f^S(t)}{\partial a_{i,j}}$ equal to the probability that X_i and Y_j are aligned, so our output is an expected alignment. Figure 6 establishes that this is not the case for the output of the Needleman-Wunsch implementation of [31].

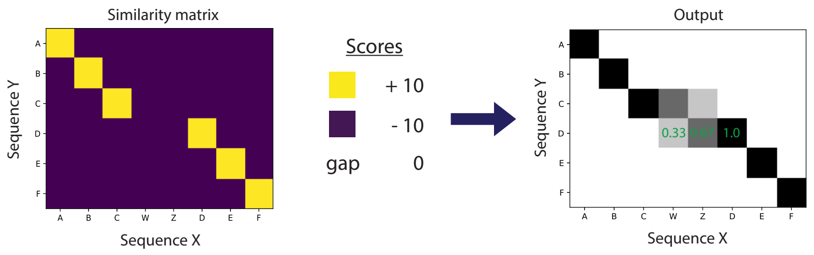

Figure 6: The output of the Needleman-Wunsch implementation of [31] is not an expected alignment. It is not the case that $Y_4 = D$ is aligned with $X_4 = W$ with probability 0.33, $X_5 = Z$ with probability 0.67, and $X_6 = D$ with probability 1.0 because in any alignment, Y_4 can be aligned to at most one residue of sequence X.

B.4 Vectorization in our SSW implementation

In 2007, Farrar achieved six-fold speed increase by implementing a "striped" version of Smith-Waterman that uses vectorization [15]. We also implement a "striped" version of smooth Smith-Waterman, and the vectorization speeds up our code substantially. In order to compute the final score $f^S(t)$, we iteratively compute the scores of the grid vertices $f^S(v_{i,j})$, which take as input the values $f^S(v_{i-1,j})$, $f^S(v_{i,j-1})$, and $f^S(v_{i-1,j-1})$. In a simple implementation, a `for` loop over i and j is used to compute the values $f^S(v_{i,j})$ (Figure 7a). To leverage vectorization, we instead compute the values $f^S(v_{i,j})$ along each diagonal in tandem, i.e. all (i,j) such that $i + j = d$. To implement this, we rotate the matrix that stores the values $f^S(v_{i,j})$ by 90 degrees so that each diagonal now corresponds to a row (see Figure 7b). In the rotated matrix, the values in a row d are a function of the values in rows $d-1$ and $d-2$, and therefore we can apply vectorization to quickly fill the matrix.

B.5 SSW options

Our smooth Smith-Waterman implementation has the following three additional options.

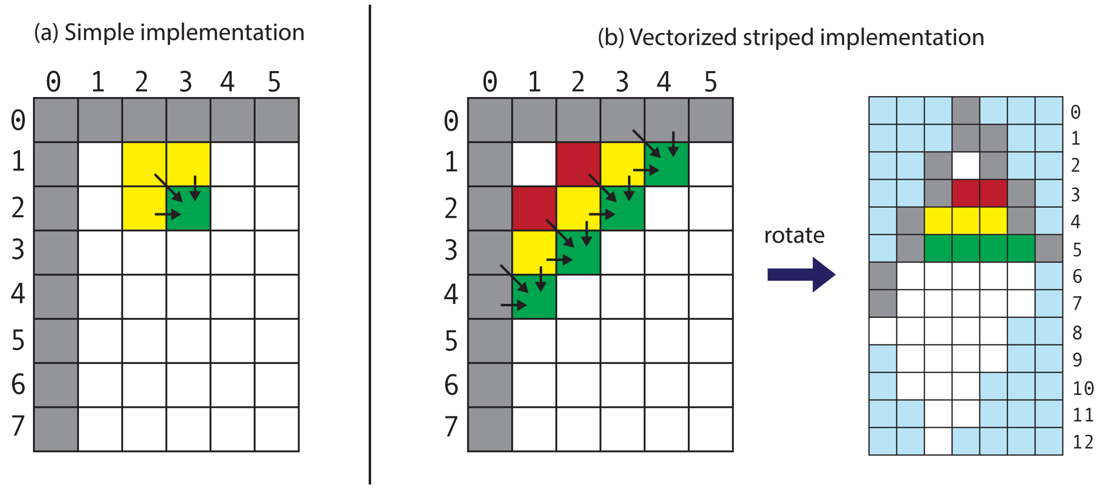
(a) Simple implementation
(b) Vectorized striped implementation
rotate
0
1
2
3
4
5
6
7
8
9
10
11
12

Figure 7: (a) In a simple implementation, the value $f^S(v_{i,j})$ are computed individually in a `for` loop over i and j. (b) In a striped implementation, that values along each diagonal in the matrix are computed in tandem. We implement this with vectorization by rotating the matrix and computing the values in each row in tandem. The blue denotes meaningless positions in the rotated matrix that we set to $-\infty$. This figure is inspired by Michael Brudno (University of Toronto).

Temperature parameter. The temperature parameter T controls the extent to which the probability distribution over alignments is concentrated on the most likely alignments; higher temperatures yield less concentrated alignments. We compute the smoothed score for the vertex v as

$$f^S(v) = T \cdot \log\left(\sum_{u \in N^-(v)} \exp\left(\frac{f^S(u) + w(u \to v)}{T}\right)\right),$$

which matches Equation (1) at the default $T = 1$.

Affine gap penalty. The "affine gap" scoring scheme introduced to Smith-Waterman by [18] applies an "open" gap penalty to the first gap in a stretch of consecutive gaps and an "extend" gap penalty to each subsequent gap. The open gap penalty is usually larger than the extend penalty, thus penalizing length L gaps less severely than L separate single residue gaps.

To implement an affine gap penalty, we use a modified alignment graph with three sets of grid vertices that keep track of whether the previous pair in the alignment was a gap or a match. Edges corresponding to the first gap in a stretch are weighted with the "open" gap penalty[1]. Figure 8a illustrates the incoming edges of the three grid vertices for (i, j). Paths corresponding to alignments with x_i and y_j matched pass through v_{ij}^D, paths corresponding to alignments with a gap at x_i pass through v_{ij}^L, and paths corresponding to alignments with a gap at y_j pass through v_{ij}^T. Storing three sets of grid vertices requires three times the memory used by the standard version. For this reason we implemented SMURF with a standard gap penalty.

Restrict turns. Smooth Smith-Waterman is inherently biased towards alignments with an unmatched stretch of X followed directly by an unmatched stretch of Y over alignments with an equally long unmatched stretch in one sequence. Consider the example illustrated in Figure 8b

[1]By convention, we charge the open gap penalty when a gap in sequence X is proceeded by a gap in sequence Y and vice versa.

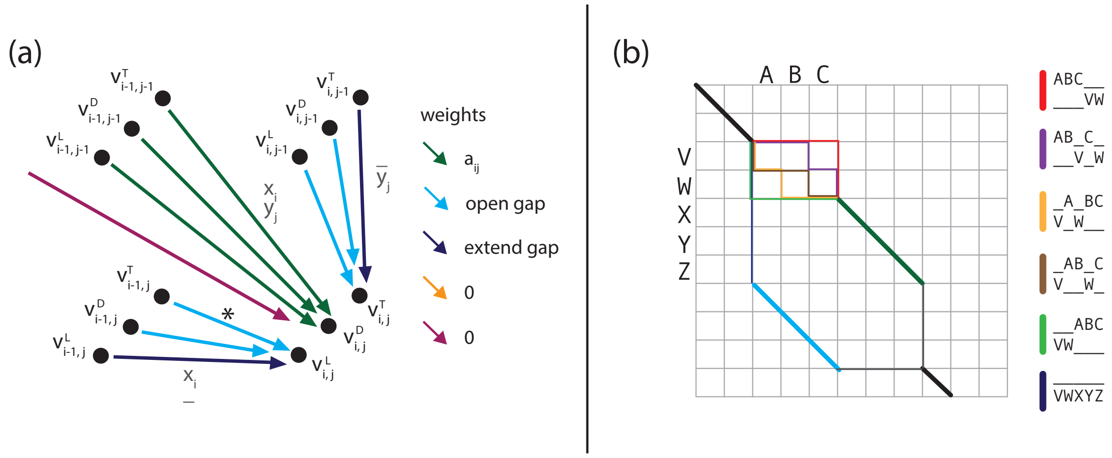

Figure 8: (a) The modification of the alignment graph from Figure 1 needed for the affine gap penalty. Incoming edges of the vertices v_{ij}^L,v_{ij}^D, and v_{ij}^T are illustrated. The colors of the edges indicate their weights. The grey labels describe the corresponding aligned pair for each group of edges. The red edge is incoming from the source vertex s. There is an outgoing edge from v_{ij}^D to the sink t for all $i, j \geq 1$ (not pictured). The edge marked with an asterisk is removed under the "restrict turns" option. (b) Without the restrict turns option, there ten paths containing both black segments and dark green segment. The red, purple, orange, brown, and light green illustrate five of these paths. There is only one path that contains both black segments and light blue segment, as depicted in navy blue. The sub-alignments corresponding to the colored segments are written on the right. With the restrict turns option the purple, orange, brown, and green paths are not valid.

where the highest scoring match states are depicted by bold black, light blue, and dark green lines. Suppose the match scores of the light blue and the dark green are identical. With a standard Smith-Waterman scoring scheme (no affine gap), the alignment containing the black and light blue segments has the same score as each alignment containing the black and dark green segments. However, there are more alignments that pass through the dark green segment. There are ten ways to align ABC and VW with no matches (the red, purple, orange, brown, and light green paths illustrate five such ways), but only one way to align $VWXYZ$ with gaps (navy blue). Smooth Smith-Waterman will assign the same probability to each of these paths. However, since ten of the eleven paths go through the dark green segment, the expected alignment output by smooth Smith-Waterman will favor the dark green segment. This bias becomes more pronounced the longer the segments; there are $\binom{L}{A}$ alignments of a sequence of length L and a sequence of length $L - A$ with no matches.

To remove this bias, we implemented "restrict turns" option that forbids unmatched stretches in the X sequence from following an unmatched stretch in the Y sequence. To do so, we again use an alignment graph with three sets of grid vertices to keep track of the previous pair in the alignment. Removing the edge with the asterisk in Figure 8a, forbids transitions from an unmatched stretch in the Y sequence to an unmatched stretch in the X sequence. When implemented with this restrict turns option, smooth Smith-Waterman will find exactly one path through the dark green and black segments in Figure 8: the path highlighted in red. Due to the increased memory requirement of the restrict turn option, we did not utilize the option in SMURF.

Global Alignment. We also implement the Needleman-Wunsch algorithm, which outputs global alignments rather than local alignments.

C SMURF

C.1 SMURF details

SMURF has two phases: BasicAlign and TrainMRF. Both begin with the learned alignment module (Figure 2), but they have different architectures and loss functions afterwards.

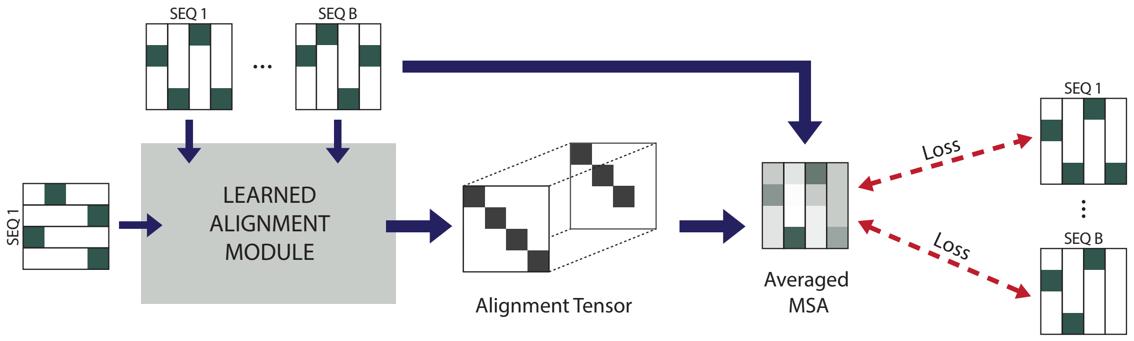

Figure 9: BasicAlign. An alignment is computed with the learned alignment module (Figure 2), and the corresponding MSA is averaged. Squared loss (Equation (4)) is computed between the averaged MSA and the one-hot encoding of the aligned input sequences.

BasicAlign. Similarity matrices produced by randomly initialized convolutions will produce chaotic alignments that are difficult for the downstream MRF to learn from. The purpose of BasicAlign is to learn initial convolutions whose induced similarity matrices yield alignments with relatively homogeneous columns (see Figure 9). The input to BasicAlign is a random subset of sequences $\mathcal{S} = \{S^{(1)}, \dots S^{(B)}\}$ in the protein family. A pairwise alignment between each sequence and the first sequence $S^{(1)}$ is produced via the learned alignment module (as described in Figure 2). This set of alignments can be viewed as an MSA where each column of the MSA corresponds to a position in the first sequence. Averaging the MSA yields the distribution of residues in each column. Let M_{ix} be the fraction of sequences in $\mathcal{S}$ with residue x aligned to position i of $S^{(1)}$,

$$M_{ix} = \frac{1}{B} \sum_{k=1}^{B} \sum_{j=1}^{\ell_k} p_{ij}^{k} \mathbb{1}\{S_j^{(k)} = x\}, \tag{3}$$

where ℓ_k is the length of $S^{(k)}$ and p_{ij}^k is the probability that position i of $S^{(1)}$ is aligned to position j of $S^{(k)}$ under the smooth Smith-Waterman alignment. The BasicAlign loss is computed by taking the squared difference between each aligned one-hot encoded sequence and the averaged MSA,

$$\mathcal{L}(\mathcal{S}, M) = \sum_{i=1}^{\ell_1} \sum_{x} \sum_{k=1}^{B} \sum_{j=1}^{\ell_k} \left(M_{ix} - p_{ij}^{k} \mathbb{1}\{S_j^{(k)} = x\} \right)^2. \tag{4}$$

TrainMRF. In TrainMRF, masked language modeling is used to learn the MRF parameters and further adjust the alignment module convolutions (see Figure 10). The input to TrainMRF is a set of sequences drawn at random from the MSA, $\mathcal{S} = \{S^{(1)}, \dots S^{(B)}\}$. A random 15% of the residues of the input sequences are masked, and the masked sequences are aligned to the query via the learned alignment module (as described in Figure 2). The parameters for the alignment module are

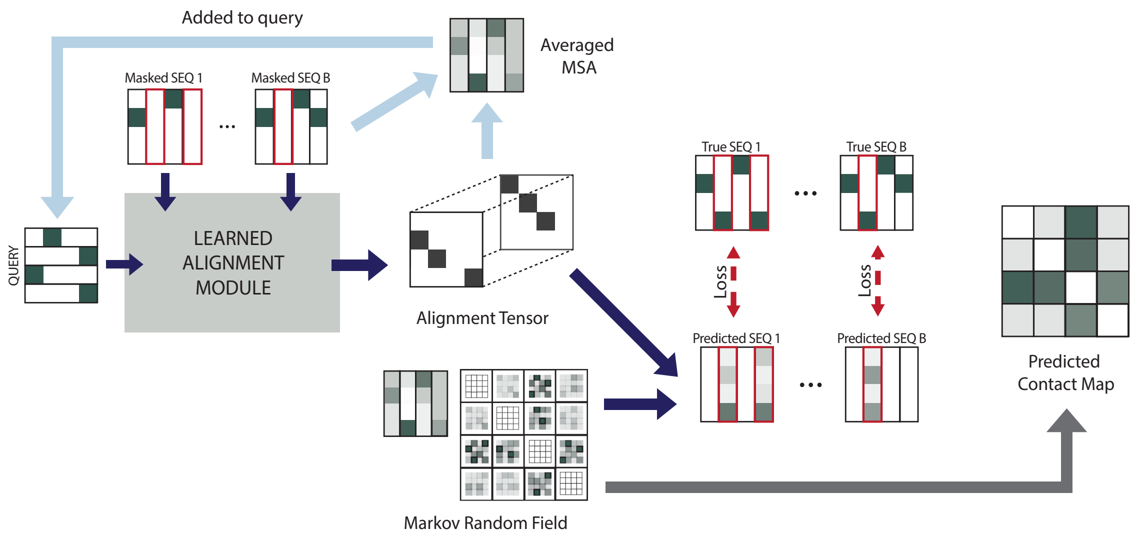

Figure 10: TrainMRF. Random positions in the input sequences are masked, then aligned with the LAM (Figure 2). A prediction for the masked positions is computed from the MRF parameters according to Equation (5). The network is trained with cross entropy loss given by Equation (6). The light blue arrows illustrate the update to the query that occurs between iterations of training; the query is a weighted average of the one-hot query sequence and a running average of the MSAs computed in previous iterations, see Equation (7). The grey arrow depicts the extraction of the contact map from the MRF matrix w at the end of training, as described in Equation (8).

initialized from BasicAlign, and the query is initialized as the one-hot encoded reference sequence for the family.

The MRF has two sets of parameters: symmetric matrices $w_{ij} \in \mathbb{R}^{A\times A}$ for $1 \leq i, j \leq \ell_R$ with $w_{ij} = w_{ji}$ that correspond to pairwise interactions of the positions in the reference sequence and position-specific bias vectors $b_i \in \mathbb{R}^A$ for $1 \leq i \leq \ell_R$. Here ℓ_R denotes the length of the reference sequence, and A is the alphabet size ($A = 21$ for amino acids and $A = 5$ for nucleotides since each includes the gap character).

After the sequences are aligned to the query, the infill distribution for each masked position is determined by the MRF parameters as follows. For a masked position j in sequence k, we define $\hat{S}_j^{(k)} \in \mathbb{R}^A$ as the predicted distribution over residues at position j of sequence $S^{(k)}$. Let p_{it}^k be the probability that position t of $S^{(k)}$ is aligned to position i of the query under the smooth Smith-Waterman alignment, and let m_t^k be the indicator that position t in sequence $S^{(k)}$ was masked. To compute $\hat{S}_j^{(k)}$, we first compute a score for each residue x that is equal to the expected value (under the smooth alignment) of the terms of the function $E(\cdot\,; b, w)$ specific to position j or involving position j and an unmasked position. Then we compute the infill distribution by taking the `softmax`. Formally,

$$\bar{S}_{jx}^{(k)} = \sum_{i=1}^{\ell_R} p_{ij}^k \left(b_{ix} + \sum_{r=1, r\neq i}^{\ell_r} \sum_{t=1}^{\ell_R} m_t^k p_{rt}^k w_{ir}\left(x, S_t^{(k)}\right) \right) \quad \text{and} \quad \hat{S}_{jx}^{(k)} = \frac{\exp\left(\bar{S}_{jx}^{(k)}\right)}{\sum_y \exp\left(\bar{S}_{jy}^{(k)}\right)}. \tag{5}$$

We train the network using a cross entropy loss and $L2$ regularization on w and b with $\lambda = .01$

$$\mathcal{L}(\mathcal{S}, p, b, w) = -\sum_{k=1}^{B}\sum_{j=1}^{\ell_R}\sum_{x} m_j^k S_{jx}^{(k)} \log \hat{S}_{jx}^{(k)} + \frac{\lambda(\ell_R - 1)(A-1)}{2}\left(\sum_{i,j}\sum_{x,y} w_{ij}(x,y)^2 + \sum_{i=1}^{\ell_r}\sum_{x} b_{ix}^2\right). \tag{6}$$

After each iteration, the query is updated to reflect the inferred MSA. Let R be the one-hot encoding of the reference sequence. We define C^{i+1} as a rolling weighted average of the MSAs learned through iteration i and Q^i as the query for iteration i,

$$C^1 = R, \qquad C^{i+1} = \eta C^i + (1-\eta)M^i, \qquad \text{and} \qquad Q^i = \gamma C^i + (1-\gamma)R \tag{7}$$

where M^i is the averaged MSA computed as described in Equation (3) from the sequences in iteration i, $\eta = 0.90$, and $\gamma = 0.3$. This process is illustrated by the light blue arrows in Figure 10.

Once training is complete, we use w to assign a contact prediction score between each pair of positions. The score c_{ij} measures the pairwise interaction between positions i and j, and $\bar{c}_{ij}$ is score after applying APC correction [11],

$$c_{ij} = \left(\sum_{x,y} w_{ij}(x,y)^2\right)^{1/2} \qquad \text{and} \qquad \bar{c}_{ij} = c_{ij} - \frac{\sum_k c_{ik}\sum_k c_{kj}}{\sum_{k,\ell} c_{k\ell}}. \tag{8}$$

C.2 Further analysis of SMURF alignments and contact predictions

RNA contact prediction. By comparing the positive predictive value (PPV) for different numbers of predicted contacts, we see that SMURF consistently yields a higher PPV for RFAM family RF00167 (Figure 3b). For RF00010, it starts off higher but then drops off faster, leading to a lower overall AUC. Upon a visual inspection of the contact predictions, MLM-GREMLIN evidently generates more false positive predictions in seemingly random locations. On the other hand, SMURF largely resolves this issue, even for RF00010, presumably as a result of a better alignment. Interestingly, SMURF's lower AUC for RF00010 can be attributed to a concentration of false positive predictions near the 5' and 3' ends. It remains unclear whether these represent a coevolution-based structural element that was not present in the specific RNA sequence deposited in PDB or whether these arise from artifacts of the learned alignment. Due to the relatively small number of RNAs with known 3D structures, we employed SMURF using the hyperparameters optimized for proteins; fine-tuning SMURF for RNA could improve performance.

Protein contact prediction and alignments. Next, we investigated the contact predictions and alignments produced by SMURF. Figure 11 and Figure 12 illustrate the contact predictions, corresponding positive predictive value (PPV) plots, and alignments for the three families that improved the most and least (respectively) under SMURF as compared to MLM-GREMLIN. The poor performance of SMURF on 3LF9A can be attributed to the misalignment of the first ≈ 25 residues of many sequences (including the one illustrated) to positions ≈ 75 to 100 of the reference rather than to the first 25 positions of the reference. This is likely because the gap penalty for leaving positions ≈ 25 to 75 unaligned outweighs the benefit of aligning to beginning of the reference. Since our code computes a local alignment, there is no penalty for leaving positions at the beginning of the reference unaligned. Perhaps using our implementation of Smith-Waterman with an affine gap penalty would lead the network to learn a less severe penalty for long gaps and arrive at correct alignment. For the most improved families, we see that SMURF tends to predict fewer false positive predictions in seemingly random positions, as observed for RNA.

Performance on smaller alignments. Finally, to test whether SMURF requires a deep alignment with many sequences, we ran SMURF on 99 protein families from [3] with at most 128 sequences. As illustrated in Figure 13, the performance of SMURF and MLM-GREMLIN are comparable even for these families with relatively few sequences.

C.3 SMURF training and hyperparameters

Throughout our hyperparameter search, we kept the following parameters constant: fraction of residues masked at 15%, number of convolution filters at 512, convolution window size at 18, regularization λ in Equation (6) at 0.01. Our hyperparameter search consisted of three stages. We initialized the gap penalty as -3 and allowed the network to learn a family-specific gap penalty.

1. First we ran a grid search with on all 190 families in the training set with learning rates $\{.05, 0.10, 0.15\}$, batch sizes $\{64, 128, 256\}$, and iterations {2000 BasicAlign /1000 TrainMRF, 3000 BasicAlign /3000 TrainMRF }. For comparison, we ran MLM-GREMLIN with the same range of learning rates and batch sizes and 3000 iterations. We found that batch size 64 and learning rate 0.05 performed best for MLM-GREMLIN.

2. Then we restricted to a smaller set of families to perform a more extensive hyperparameter search; we included the seven families where MLM-GREMLIN's AUC was less than 0.45 (3AKBA, 3AWUA, 5BY4A, 4C6SA, 3OHEA, 3ERBA, 4F01A) and six families where SMURF consistently performed substantially worse than MLM-GREMLIN (1NNHA, 3AGYA, 4LXQA, 1COJA, 2D4XA, 4ONWA). We considered the following hyperparameter options: learning rates $\{.05, 0.10\}$, batch sizes $\{64, 128, 256\}$, iterations {2000 BasicAlign /1000 TrainMRF, 2000 BasicAlign /2000 TrainMRF, 3000 BasicAlign /1000 TrainMRF }, MSA memory fraction $\eta \in \{0.90, 0.95\}$, and MSA query fraction $\gamma \in \{0.3, 0.5, 0.7\}$.

3. Based on the results of the above hyperparameter search on the select families, we performed a final hyperparameter search on the entire training set. We noticed that performance was better for larger batch sizes, but it was not always possible to run the large batch sizes on our 32 GB GPU for families with longer sequences. For our final hyperparameter search, we used the largest batch size of $\{64, 128, 256\}$ that would fit in memory for each family. We set $\eta = 0.90$, $\gamma = 0.3$, and selected 3000 BasicAlign /1000 TrainMRF iterations because these parameters lead to relatively strong results across the restricted set of families. Learning rate 0.10 outperformed learning rate 0.05 on the restricted set, but learning rate 0.05 generally outperformed learning rate 0.10 in the initial grid search on the full training set. We ran a final test with the aforementioned parameters and the two learning rates on the entire training set, and found that learning rate 0.05 was optimal overall.

   We also ran 4000 iterations of MLM-GREMLIN with predetermined optimal parameters: learning rate 0.05 and batch size 64. We found very similar performance between 3000 and 4000 iterations of MLM-GREMLIN. We chose to compare SMURF to 4000 iterations of MLM-GREMLIN so that both methods were trained for 4000 iterations.

C.4 Ablation in SMURF

We performed an ablation study in which we removed three features of SMURF (Figure 14).

First we replaced smooth Smith-Waterman with a differentiable "pseudo-alignment" procedure, similar to [5]. Instead of applying smooth Smith-Waterman to the similarity matrix, we computed a pseudo-alignment by taking the softmax of the similarity matrix row-wise and column-wise, multiplying the resultant matrices, and taking the square root. This technique performed far worse than SMURF.

Next, we consider the roll of BasicAlign. We found that skipping BasicAlign and running TrainMRF for 4000 iterations degraded performance, thus indicating the importance of the inital convolutions found in BasicAlign.

Finally, we found that changing the query between iterations did not improve results. Preliminary results on the training set had suggested that changing the query improved results for some families. Further investigation is needed to determine the benefits changing the query between iterations.

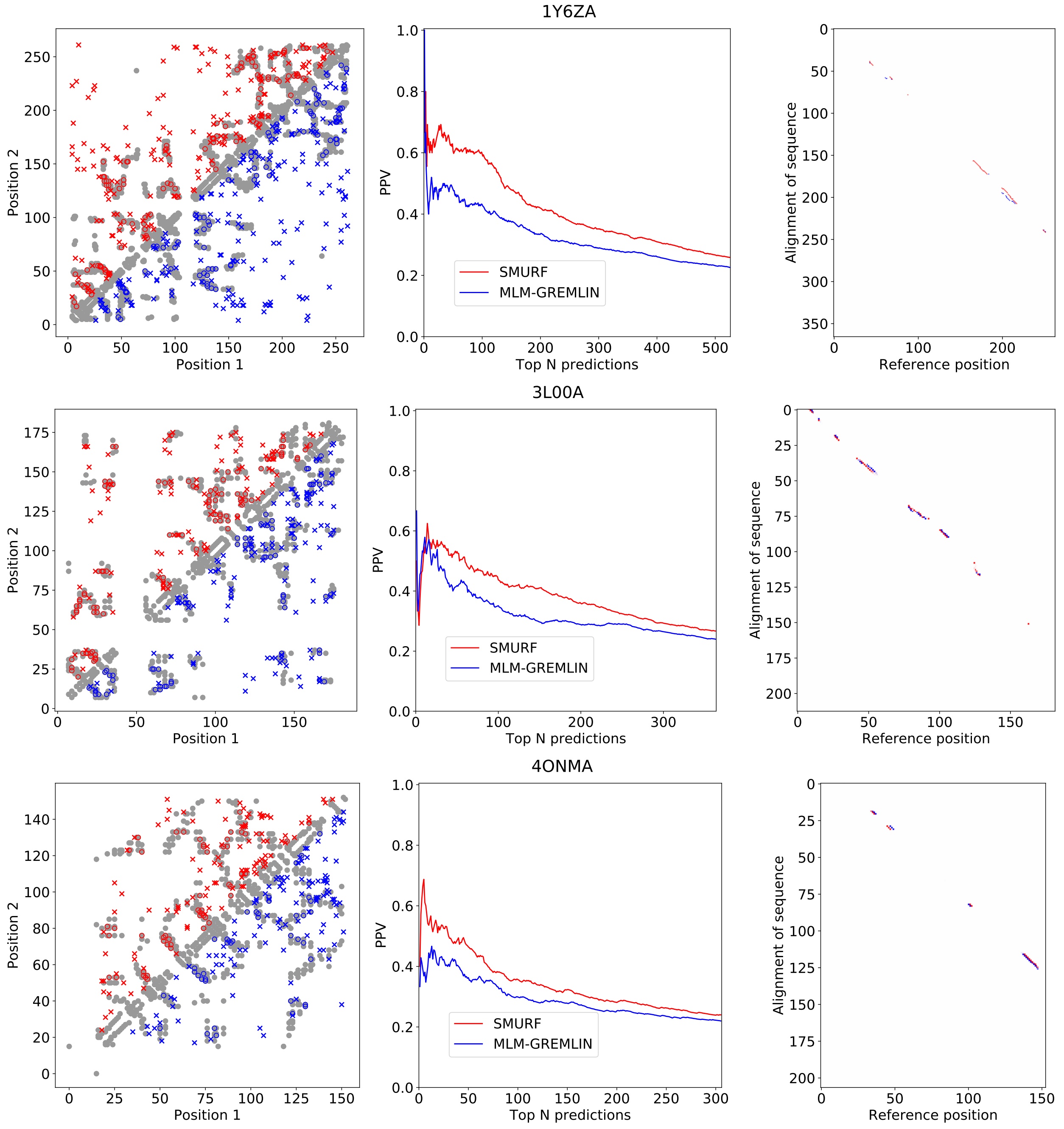

Figure 11: Three most improved protein families. Left: Comparison of contact predictions between SMURF (red) and MLM-GREMLIN (blue). Gray dots represent PDB-derived contacts, circles represent a true positive prediction, and x represents a false positive prediction. Middle: The positive predictive value (PPV) for different numbers of top N predicted contacts, with N ranging from 0 to $2L$. Right: Comparison of the alignment of a random sequence in the family to the reference sequence. Red indicates aligned pairs that appear in the SMURF alignment, but do not appear in the given alignment. Blue indicate aligned pairs that appear in the given alignment, but do not appear the alignment found by SMURF.

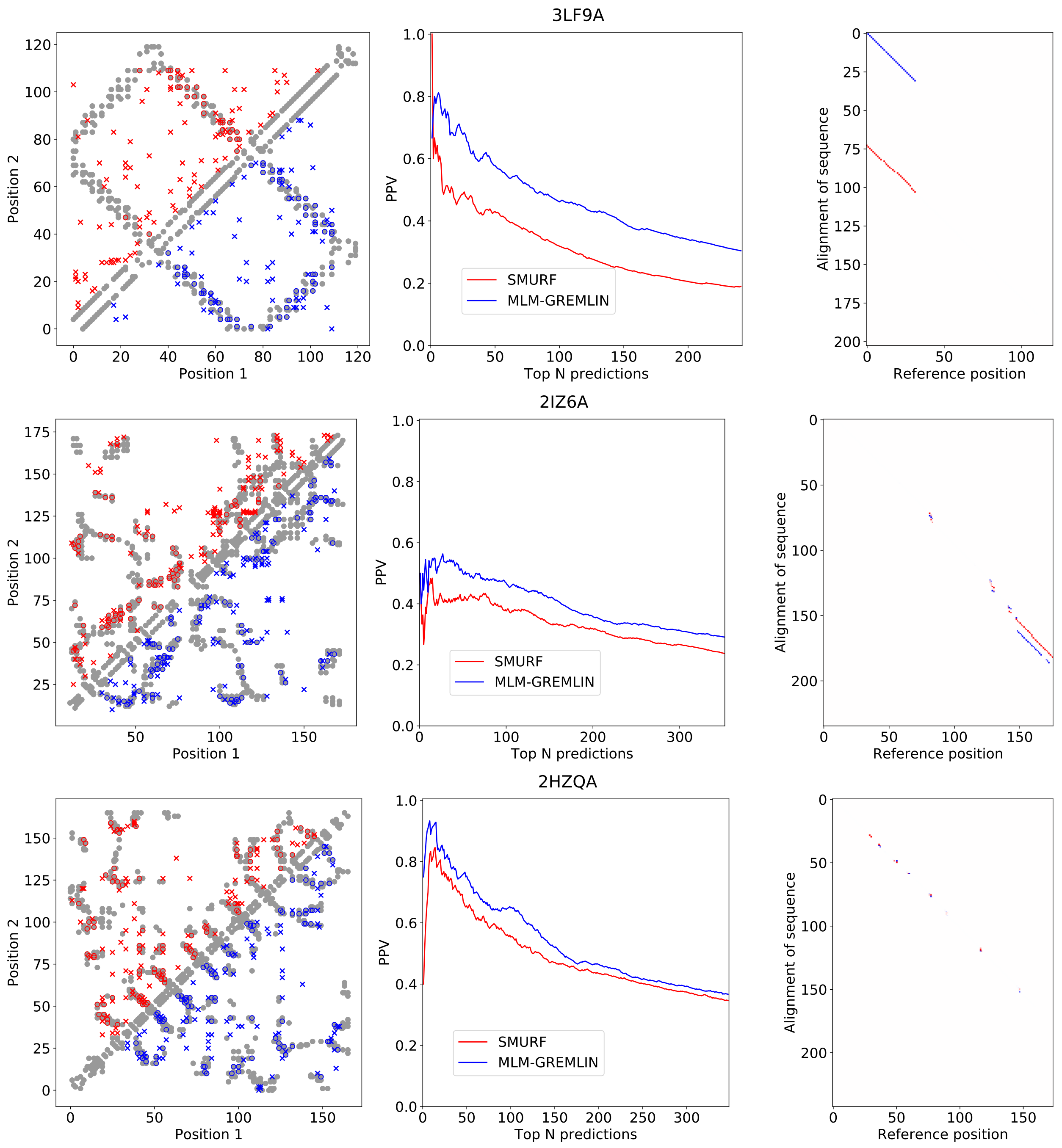

Figure 12: Three worst performing protein families (as compared to MLM-GREMLIN). Left: Comparison of contact predictions between SMURF (red) and MLM-GREMLIN (blue). Gray dots represent PDB-derived contacts, circles represent a true positive prediction, and x represents a false positive prediction. Middle: The positive predictive value (PPV) for different numbers of top N predicted contacts, with N ranging from 0 to $2L$. Right: Comparison of the alignment of a random sequence in the family to the reference sequence. Red indicates aligned pairs that appear in the SMURF alignment, but do not appear in the given alignment. Blue indicate aligned pairs that appear in the given alignment, but do not appear the alignment found by SMURF.

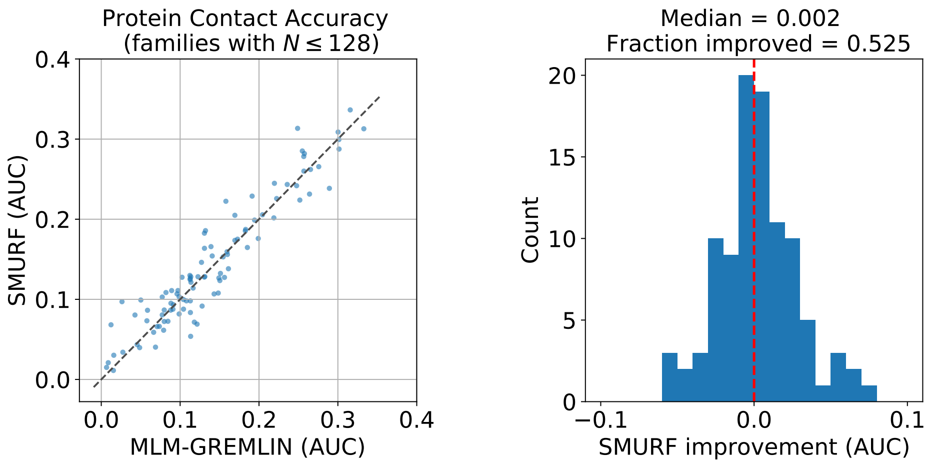

Figure 13: Performance comparison for families with at most 128 sequences. Left: Scatter plot of the AUC of the top L predicted contacts for SMURF versus MLM-GREMLIN. Right: Histogram of the difference in AUC between SMURF and MLM-GREMLIN.

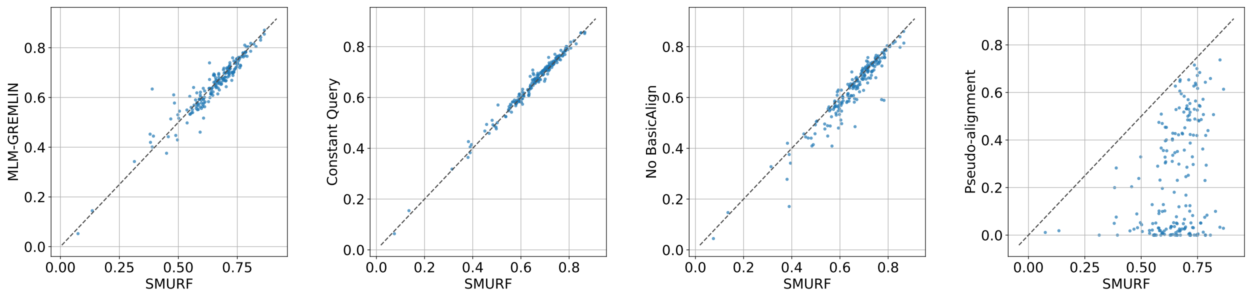

Figure 14: Contact AUC for SMURF versus ablated methods. Each point represents one family in the test set.

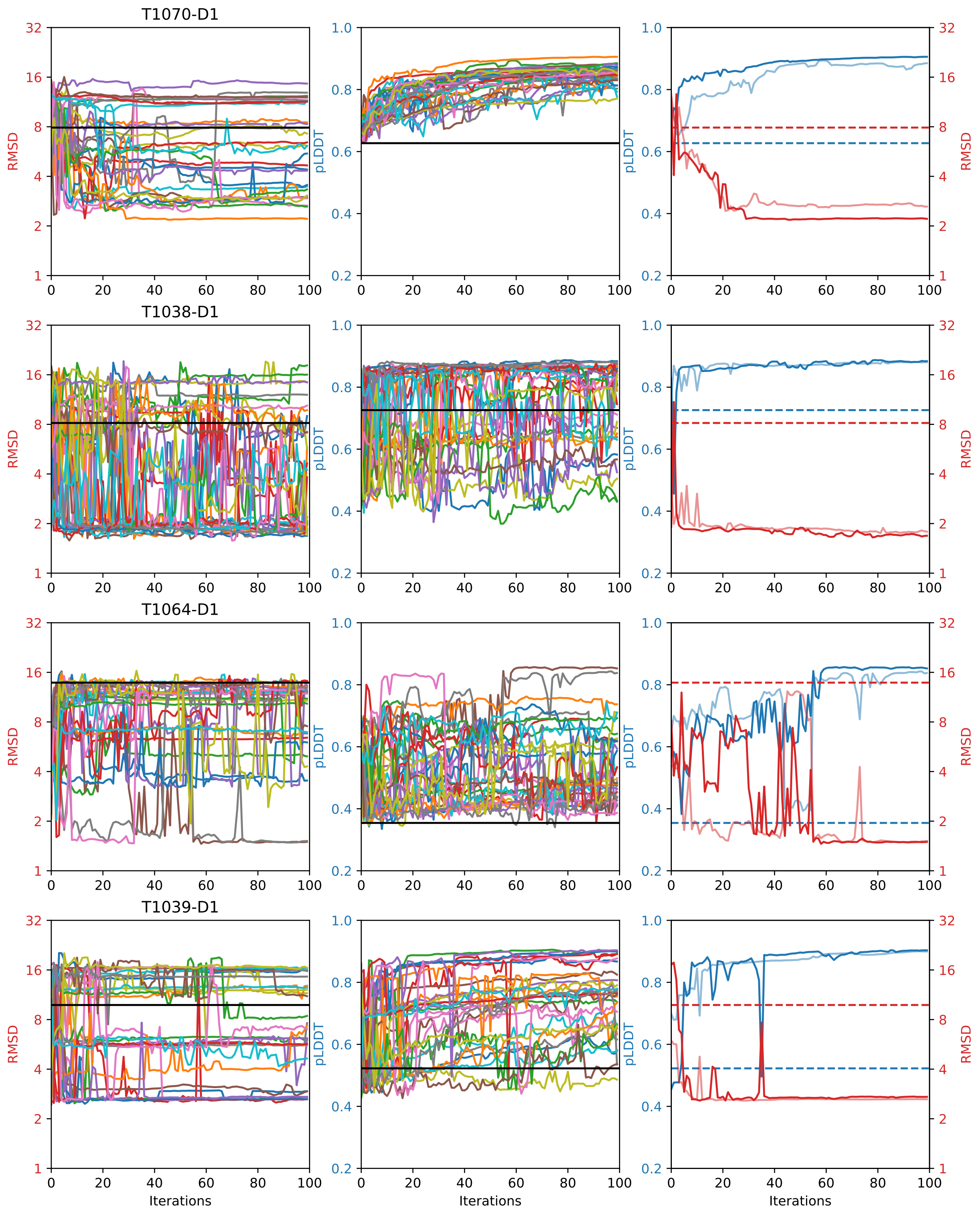

Figure 15: Optimization trajectories for CASP targets in Figure 4. The First column is the RMSD (distance to ground truth) monitored during optimization. The second column is the predicted LDDT (confidence metric) being maximized. The third column highlights the top 2 trajectories select by best terminal pLDDT. The red lines are the RMSD and blue lines are the pLDDT.

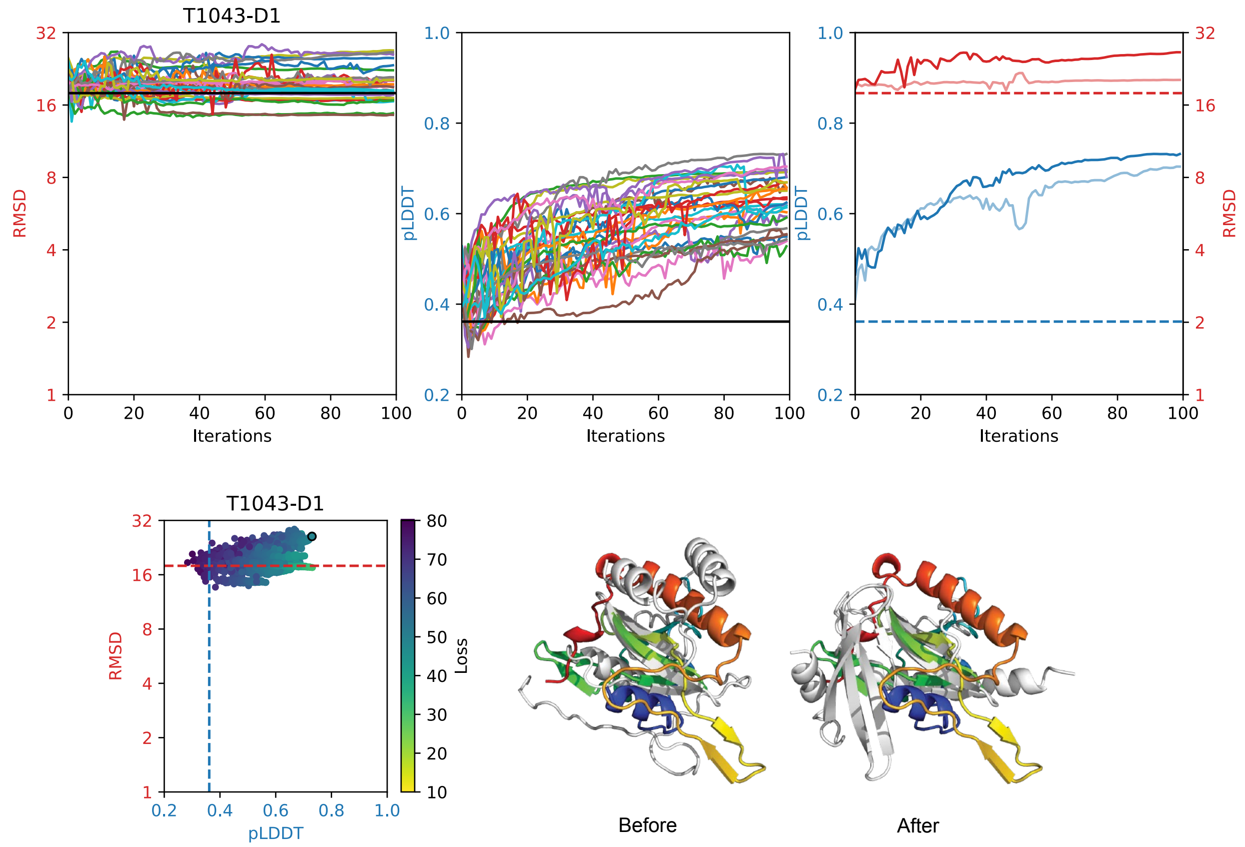

Figure 16: Optimization trajectories for CASP target T1043-D1. the scatter plot shows the sampled pLDDT and RMSD across all trajectories. The dotted lines show the initial pLDDT and RMSD (root-mean-squared-distance to native structure) using the MSA from MMseqs2. The native structure is rainbow colored, and the predictions are overlaid and colored white. After maximizing the confidence metric, the structure with the max pLDDT (circled) is shown in the after column. For other targets see Figure 4.