HHS Public Access

Author manuscript

Nat Mach Intell. Author manuscript; available in PMC 2022 August 12.

Published in final edited form as:

Nat Mach Intell. 2022 January ; 4(1): 41–54. doi:10.1038/s42256-021-00428-6.

Interpreting Neural Networks for Biological Sequences by Learning Stochastic Masks

Johannes Linder[1,*], **Alyssa La Fleur**[1], **Zibo Chen**[2], **Ajasja Ljubeti**[2], **David Baker**[2], **Sreeram Kannan**[3], **Georg Seelig**[1,3]

[1]Paul G. Allen School of Computer Science and Engineering, University of Washington

[2]Institute for Protein Design, University of Washington

[3]Department of Electrical and Computer Engineering, University of Washington

Abstract

Sequence-based neural networks can learn to make accurate predictions from large biological datasets, but model interpretation remains challenging. Many existing feature attribution methods are optimized for continuous rather than discrete input patterns and assess individual feature importance in isolation, making them ill-suited for interpreting non-linear interactions in molecular sequences. Building on work in computer vision and natural language processing, we developed an approach based on deep learning - Scrambler networks - wherein the most salient sequence positions are identified with learned input masks. Scramblers learn to predict Position-Specific Scoring Matrices (*PSSMs*) where unimportant nucleotides or residues are scrambled by raising their entropy. We apply Scramblers to interpret the effects of genetic variants, uncover non-linear interactions between cis-regulatory elements, explain binding specificity for protein-protein interactions, and identify structural determinants of *de novo* designed proteins. We show that Scramblers enable efficient attribution across large datasets and result in high-quality explanations, often outperforming state-of-the-art methods.

Deep Neural Networks have successfully been applied to a diverse set of biological sequence prediction problems, including predicting transcription factor binding [1, 2, 3, 4], chromatin modification and accessibility [5], RNA processing [6, 7, 8, 9], translation regulation [10] and protein structure [11, 12]. Neural networks excel at learning complex relationships from large datasets without requiring much tuning, but interpreting their predictions is challenging because the learned regulatory logic is embedded deep within the network layers. Nevertheless, interpretability is necessary to connect network predictions

[*]Correspondence to: jlinder2@cs.washington.edu.

Author Contributions

J.L., A.L., S.K and G.S. conceived and developed the project. J.L. and A.L. performed the computational analyses with input from A.Lj. J.L., A.L., S.K and G.S. wrote the paper with input from A.Lj., Z.C. and D.B.

Code Availability

All code [68] is available at http://www.github.com/johli/scrambler (DOI: 10.5281/zenodo.5676173), licensed under MIT License. External software used in this study is listed in the Methods section.

Competing Interests Statement

The authors declare no competing interests.

to established biology, learn new regulatory rules or validate sequence designs [13, 14, 15, 16, 17, 18, 19, 20, 21].

Nonlinear interactions are abundant between biological sequence features. Such dependencies make it necessary to consider groups of features and their surrounding context when interpreting a model prediction instead of independently evaluating each feature in isolation. However, many current neural network attribution methods rely on local perturbations, limiting their ability to identify such relationships. For example, 5' UTRs can contain multiple upstream start codons (uAUGs) [22, 23, 24] preceding the primary start. Which AUG is chosen as the start of translation depends on the extent to which the surrounding context – e.g., competing AUGs – represses its usage. Two nearby uAUGs may even "hide" each other, as each AUG is capable of repressing translation initiation at the primary start independently. In-silico saturation mutagenesis – which systematically exchanges one nucleotide and approximates its importance by predicted functional change – would incorrectly conclude both uAUGs are irrelevant: Neither uAUG would be identified as repressive, as knocking down only one would not change the prediction. Other local approximation methods face similar problems, such as those basing their estimation on gradients [25, 26, 27, 28, 29, 30] or local linear models [31].

In this paper, we introduce a feature attribution approach tailored for biological sequence predictors and based on previous work in learning interpretable input masks [32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. We train a deep neural network, referred to as the Scrambler, to learn masks that include only the smallest set of features in an input sequence necessary to reconstruct the original prediction. A mask that fulfills the inclusion objective overcomes the issues with local approximation mentioned above. For example, such a mask would learn to include one of the uAUGs from the 5' UTR example to preserve the repressive prediction. In a complementary formulation, we optimize masks to occlude the smallest set of features that destroy the prediction. These features may overlap with those identified by inclusion masks but are not necessarily identical; for the 5' UTR example, the occlusion mask would need to occlude both of the uAUGS to destroy the repressive prediction. In general, for features involved in an OR-relationship, inclusion would identify one feature while occlusion would identify all. Conversely, for AND-like features, inclusion would identify all features and occlusion just one.

The earliest mask-based attribution methods masked inputs by either fading or blurring [32, 34], which is ill-suited for one-hot encoded sequence patterns. More recent methods replace masked input features with random samples or counterfactual values to keep masked inputs in the distribution of valid predictor patterns [37, 39]. However, these recent methods are based on individually optimizing each mask rather than learning a parametric model. Here, we combine ideas from deep learning with probabilistic masking to interpret biological sequences. Specifically, Scramblers learn to output Position-Specific Scoring Matrices (*PSSMs*) with minimal (inclusion) or maximal (occlusion) conservation such that discrete samples either reconstruct or destroy the prediction. A sequence position is 'masked' by raising the entropy, or *temperature*, of the underlying feature distribution such that samples sent to the predictor become less informative, and the Scrambler is trained by backpropagation using a gradient estimator.

We show that Scramblers avoid overfitting to spurious per-example signals, which can be problematic for model-free methods. Furthermore, using a parametric masking model allows us to interpret new patterns more efficiently than per-example masking methods. Scramblers are also conceptually different from methods such as L2X [35] and INVASE [36], which mask input patterns by replacing features with zeros while simultaneously training a new ad-hoc predictor model that attempts to reconstruct the original predictions. Re-training the predictor is required since the zero-based masks push the patterns out-of-distribution. In contrast, Scramblers directly reconstruct the original predictor model without re-training, and keep patterns in distribution with a sample-based mask. This allows us to efficiently train the Scrambler on a smaller, or more narrowly defined, dataset than the original predictor and still obtain high-quality interpretations. We show in our benchmarks that L2X and INVASE produce lower-quality interpretations than Scramblers when trained on reduced datasets ranging in size from 10, 000-40, 000 sequences, and they converge poorly when trained on the full data. See Supp. Table 1 for a summary comparison of all masking methods.

In addition to introducing probabilistic masks to biological sequences, we develop several improvements for mask-based interpretation. For example, Scramblers can discover multiple salient feature sets within a single input pattern using mask dropout- and bias layers, enabling users to specify features to include or exclude in the solution at inference time. In addition to finding reconstructive features, we alternatively optimize the Scrambler to find enhancing or repressive features that either maximize or minimize the prediction. We also explore different architectures for interpreting pairwise predictors in the context of protein-protein interaction. Finally, to reduce interpretation artifacts for predictors where input patterns contain a highly variable number of features, we apply per-example optimization as a fine-tuning step to the initial Scrambler outputs. Throughout the paper, we show that Scramblers learn meaningful feature attributions for several visual- and sequence-predictive tasks, including RNA regulation, protein-protein interaction, and protein tertiary structure, often outperforming other interpretation methods.

Results

Scrambling Networks

The Scrambler architecture is illustrated in Fig. 1a (left). Given a differentiable pre-trained predictor $\mathscr{P}$ and a one-hot encoded input pattern $\boldsymbol{x} \in \{0, 1\}^{N \times M}$ (representing an N-length sequence), we let a trainable network $\mathscr{S}$ called the *Scrambler* predict a set of real-valued importance scores $\mathscr{S}(\boldsymbol{x}) \in (0, \infty]^N$. These scores are used in Eq. 1 below to produce a probability distribution which interpolates between the input pattern $\boldsymbol{x}$ and a non-informative background distribution $\tilde{\boldsymbol{b}} \in \mathbb{R}^{N \times M}$ (Fig. 1b-c).

$$\hat{\boldsymbol{x}}_{\mathscr{S}} = \sigma\left(\log \tilde{\boldsymbol{b}} + \boldsymbol{x} \times \dot{\mathscr{S}}(\boldsymbol{x})\right) \tag{1}$$

Here, σ denotes the softmax $\sigma(\boldsymbol{l})_{ij} = \frac{e^{\boldsymbol{l}_{ij}}}{\sum_{k=1}^{M} e^{\boldsymbol{l}_{ik}}}$ and $\dot{\mathcal{S}}(\boldsymbol{x}) \in (0, \infty]^{N \times M}$ represent the importance scores $\mathcal{S}(\boldsymbol{x})$ which have been broadcasted at position i to all channels j. In all experiments, the Scrambler $\mathcal{S}$ is a residual network of dilated convolutions (Extended Data Fig. 1a-c) [42]. The output $\hat{\boldsymbol{x}}_{\mathcal{S}}$ becomes a parameterization of a probability distribution of the input. Specifically, $\hat{\boldsymbol{x}}_{\mathcal{S}}$ is a set of N categorical softmax-nodes, or a PSSM. The role of $\tilde{\boldsymbol{b}}$ is to keep samples from this PSSM in distribution and along the manifold of valid patterns, and is here taken as the mean input pattern across the training set (Eq. 10). When $\mathcal{S}(\boldsymbol{x})_i$ is close to 0, $\hat{\boldsymbol{x}}_{\mathcal{S},i}$ becomes $\tilde{\boldsymbol{b}}_i$ (the background distribution) and when $\mathcal{S}(\boldsymbol{x})_i$ is close to ∞, $\hat{\boldsymbol{x}}_{\mathcal{S},i}$ becomes $\boldsymbol{x}_i$ (the original input). $\mathcal{S}(\boldsymbol{x})_i$ thus defines the inverse feature sampling temperature at position i in the PSSM.

K discrete samples $\boldsymbol{x}_{\mathcal{S}}^{(k)}$ drawn from $\hat{\boldsymbol{x}}_{\mathcal{S}}$ are passed to the predictor $\mathcal{P}$ and gradients are backpropagated to $\mathcal{S}$ using either Softmax Straight-Through estimation [43] or the Gumbel distribution [44]. By comparing the predictions $\mathcal{P}(\boldsymbol{x}_{\mathcal{S}}^{(k)})$ of the scrambled input samples to the original prediction $\mathcal{P}(\boldsymbol{x})$, we train the Scrambler $\mathcal{S}$ to minimize a predictive reconstruction error subject to a conservation penalty which enforces high entropy. We refer to this formulation as the *Inclusion*-Scrambler, as it must learn to include features to reconstruct the original prediction while maintaining high entropy of $\hat{\boldsymbol{x}}_{\mathcal{S}}$. Alternatively, we can train $\mathcal{S}$ to find the smallest set of features in $\boldsymbol{x}$ to randomize (i.e., maximizing the conservation of $\hat{\boldsymbol{x}}_{\mathcal{S}}$) to maximally perturb $\mathcal{P}(\boldsymbol{x}_{\mathcal{S}}^{(k)})$ from $\mathcal{P}(\boldsymbol{x})$ (Fig. 1a, right). We refer to this inverse formulation as the *Occlusion-Scrambler*. For both formulations, we define the reconstruction error as the KL-divergence $\mathrm{KL}[\mathcal{P}(\boldsymbol{x}_{\mathcal{S}}^{(k)}) \| \mathcal{P}(\boldsymbol{x})]$ between scrambled and original predictions. To minimize the conservation of $\hat{\boldsymbol{x}}_{\mathcal{S}}$ while still keeping samples in distribution, we optimize the KL-divergence $\mathrm{KL}[\tilde{\boldsymbol{b}} \| \hat{\boldsymbol{x}}_{\mathcal{S}}]$ between $\hat{\boldsymbol{x}}_{\mathcal{S}}$ and the background distribution $\tilde{\boldsymbol{b}}$. We control the expected entropy by fitting $\mathrm{KL}[\tilde{\boldsymbol{b}} \| \hat{\boldsymbol{x}}_{\mathcal{S}}]$ to a target conservation value t_{bits} rather than minimizing or maximizing it unbounded. The full training objective for the Inclusion-Scrambler is given in Eq. 2.

$$\min_{\mathcal{S}} \left(\frac{1}{K} \sum_{k=1}^{K} \mathrm{KL}[\mathcal{P}(\boldsymbol{x}_{\mathcal{S}}^{(k)}) \| \mathcal{P}(\boldsymbol{x})] \right) + \lambda \cdot \left(t_{\mathrm{bits}} - \frac{1}{N} \cdot \mathrm{KL}[\tilde{\boldsymbol{b}} \| \hat{\boldsymbol{x}}_{\mathcal{S}}] \right)^2 \tag{2}$$

We compare several different attribution methods in our experiments. The baseline method used for comparison, *Perturb*, exchanges the categorical value of one letter or pixel at a time and estimates the absolute value in predicted change as the importance score. Comparisons are made against Perturb (baseline), Gradient Saliency [25], Guided Backprop [27], Integrated Gradients [28], DeepLIFT [29] (using RevealCancel for MNIST and Rescale from DeepExplain [45] for the remaining tasks), SHAP DeepExplainer [30], the preservation/perturbation methods of Fong et al. (TorchRay) [33], Dabkowski et al. (Saliency Model) [34] and Carter et al. ('SIS') [39] and, finally, the feature selection methods L2X [35] and INVASE [36]. For comparison, we also test a version of the

Scrambler with a zero-based masking operator (referred to as Zero Scrambler). See Methods for a detailed description of how each method was used and which task(s) were tested. Note that for methods that learn ad-hoc interpreter models (e.g., Scramblers, L2X, INVASE), we train the ad-hoc models on a reduced data set of ~ 10, 000-40, 000 examples (depending on the task), in contrast to the 100, 000s to millions of examples the original predictor models were trained on. However, in supplemental analyses we also attempted to train L2X and INVASE on the full data.

Finding Salient Pixels in MNIST Images

We first used the Scrambler to visualize important regions in binarized images of the two MNIST digits '3' and '5' (Fig. 2a). The predictor was a convolutional neural network (CNN) [46] trained to discriminate between all ten digits. We trained one Inclusion-Scrambler ($t_{\text{bits}} = 0.005$, $\lambda = 500$) and one Occlusion Scrambler ($t_{\text{bits}} = 0.3$, $\lambda = 500$). We performed two benchmark comparisons to other attribution methods (Fig. 2b, Extended Data Fig. 2a): first, based on the importance scores of each method, we replaced all but the top X% most important pixels with random samples from the background and measured the KL-divergence between the predictions of the original and randomized test set images. Next, we inversely tested how well these top-ranked features perturbed the prediction by replacing only the top X% pixels with random samples. The Scramblers were superior in each benchmark, as the median KL-divergence was the lowest and highest, respectively. In addition to the upper portions of the digits themselves, the Scramblers identified open regions adjacent to the digits as important, indicating that the classifier relies on both foreground and background pixels when making its prediction. For example, the open background region in the top left quadrant of digit '3' is a unique distinguishing feature shared among all examples of '3'. These salient 'background' features are identified by methods such as DeepLIFT and DeepSHAP, but not by other methods that only identify foreground pixels.

Identifying Alternate Feature Sets with Dropout and Bias Layers—There are often multiple feature sets a classifier can use within a single input pattern to make its prediction. Scramblers can be trained to select these alternate salient feature sets dynamically using a dropout pattern $\boldsymbol{D}$ which is used to mask some positions in the predicted scores $\mathcal{S}(\boldsymbol{x})$. Similarly, bias patterns $\boldsymbol{T}$ are used to encode specific positions to be included in the solution. Importantly, the Scrambler $\mathcal{S}$ also receives $\boldsymbol{D}$ and $\boldsymbol{T}$ as additional input, allowing the network to learn to output alternate (or additional) scores conditioned on which positions were dropped or enforced (Fig. 2c, Extended Data Fig. 2b-d; see Methods for details). During training, we sample random dropout- and bias patterns. At attribution time, we supply specifically crafted patterns to exclude or enforce in the predicted solution to enable targeted feature attributions.

For the MNIST classifier, bias patterns allow us to explore features beyond the initial salient pixels found by default (Fig. 2d). Conversely, we can use dropout patterns to focus attribution to pixels in a subsection of an image to discover new salient features (Fig. 2e). In Extended Data Fig. 2e-f, we trained a Scrambler with tunable divergence against the

background $\tilde{\boldsymbol{b}}$, demonstrating how the entropy of the solution can be set dynamically at attribution time, thus providing solutions at multiple scales of t_{bits} with a single network.

The cis-Regulatory Code of Alternative Polyadenylation

Next, we used Scramblers to interpret regulatory sequence elements in the 3' untranslated region (UTR) of pre-mRNA (Fig. 3a). Specifically, using APARENT [7] – a CNN capable of predicting cleavage and polyadenylation from primary sequence – we trained an Inclusion-Scrambler to reconstruct isoform predictions for a subset of the original APARENT training data ($t_{\text{bits}} = 0.25$, $\lambda = 1$). Since we anticipated important polyadenylation features like RNA binding protein (RBP) motifs to consist of short subsequences, we regularized the Scrambler by fixing the final layer of the network to a Gaussian filter to encourage the selection of contiguous nucleotides for masking (reminiscent of a technique proposed by Fong et al. [33]). We found that the Inclusion-Scrambler learned to recognize known regulatory binding factors associated with alternative polyadenylation (Extended Data Fig. 3a), such as CFIm, CstF, HNRNPL, ENOX1, PABPN1, HuR, and RBM4 [47, 48, 49, 50]. When re-training the Scrambler under the Occlusion objective, the network learned patterns that resulted in more extreme perturbations without the fixed Gaussian filter than with it (Extended Data Fig. 3b-c). However, when inspecting these patterns, they clearly exploited the importance of scattered T's in favor of knocking out biologically relevant motifs (Extended Data Fig. 3d).

The Scrambler outperformed other attribution methods when reconstructing the isoform predictions of the APARENT model using only the top-ranked nucleotides and replacing the remaining positions with random letters (Fig. 3b, Extended Data Fig. 3e-f; Median KL = 0.32 when keeping the 5% most important features). We next tested how well the discovered features generalized to other APA-predictive models. To this end, we again perturbed the sequences by keeping only the 5% most important nucleotides when interpreting APARENT, but we now tested how well these features reconstructed predictions made by DeeReCT-APA, a model trained on Mouse 3'-end sequencing data [51] (Extended Data Fig. 3f). The Scrambler reconstructed the DeeReCT-APA predictions the most (Spearman r = 0.67 when keeping the 5% most important features, $n = 1,737$), but was closely followed by DeepLIFT (r = 0.65) and Integrated Gradients (r = 0.66). Finally, we trained a deeper version of the Scrambler with 20 residual blocks instead of 4 blocks. While the deeper Scrambler reconstructed APARENT predictions even better (Median KL = 0.28), the performance marginally decreased on DeeReCT-APA (r = 0.65) (Extended Data Fig. 3f).

Interpreting Genetic Variants of APA—We can interpret the functional impact of mutations by running the Scrambler on wild-type and variant human polyadenylation signals. For example, the Scrambler correctly detects the creation of a cryptic CstF binding site caused by a deleterious variant (*108C>T*) in the *F2* (Prothrombin) gene, which is known to cause or contribute to Thrombophilia due to increased polyadenylation efficiency (Fig. 3c) [52, 53, 54]. The detected CstF motif is qualitatively better separated from the Scrambler sequence logo's background compared to Perturbation. Furthermore, the Scrambler appears highly sensitive in detecting loss of RBP binding motifs due to single nucleotide variants for several disease-associated human polyadenylation signals [55, 56, 57, 58] (Extended Data Fig. 3g).

Reconstructive, Enhancing and Repressive Motifs—Interestingly, with as little as 0.25 out of 2.0 target bits of mean conservation in the scrambled sequences, there was a very high correlation between scrambled and original sequence predictions from the APARENT test set ($R^2 = 0.85$; Fig. 3d). In other words, a small set of regulatory features in each polyadenylation signal explained most of the variation. Next, we altered the Scrambler training objective to find the smallest set of features that maximize or minimize a prediction rather than reconstructing it (see Methods for details). We found that the Scrambler could partition cis-regulatory elements within polyadenylation signals into enhancing (e.g., CFIm, HuR, Cstf) and repressive motifs (e.g., ENOX1, RBM4, PABPN1), in agreement with the known function for these motifs and associated RBPs (Fig. 3e, Extended Data Fig. 3h).

The Rules of Translation Efficiency in the 5' UTR

The translation efficiency of mRNA is controlled by complex regulatory logic in its 5' UTR. For example, an in-frame (IF) start and stop codon create an IF upstream open reading frame (uORF), which typically represses translation, while neither an IF upstream start or upstream stop codon individually represses translation (NAND logic). Sequences with multiple IF starts and stops can further complicate translational logic by creating NAND-OR hybrid functions with overlapping IF uORFs.

We trained an Inclusion-Scrambler to reconstruct the mean ribosome load (MRL – a proxy for translation efficiency) of 10, 000 5' UTRs as predicted by the CNN Optimus 5-Prime (Fig. 4a) [10]. To test how well attribution methods detect multiple regulatory elements, we generated synthetic 5' UTR test sets with one (1 IF start, 1 IF stop), two (1 IF start/2 IF stops or 2 IF starts/1 IF stop), or four (2 IF starts/2 IF stops) overlapping IF uORFs. We then measured to what extent the most important nucleotides of each method corresponded to the IF start and stop positions (Fig. 4b, Extended Data Fig. 4a-b). In the simplest case of a single start and stop, nearly all methods had high accuracy for recovering these elements. However, as the number of IF uORFs increased, the Scrambler often outperformed other methods considerably. An example 5' UTR with multiple IF uORFs is shown in Fig. 4c; here, the multiple stops effectively hide one another from many local attribution methods. Furthermore, L2X, which trains a new predictor model simultaneously as the interpreter is learned, overfits by only selecting one of the nucleotides in each of the 'ATG' and 'TAG' motifs (hence the low accuracy). The results did not improve for L2X even when trained on the full 5' UTR dataset (Extended Data Fig. 4c).

Interpreting Genetic Variants in the 5' UTR—The combinatorial nature of out-of-frame start codons (OOF uAUGs) and IF uORFs makes variant interpretation in human 5' UTRs complicated [22, 23, 24]. In Fig. 4d, we interpret a variant, *rs1160558441*, which creates an OOF uAUG that Optimus 5-Prime predicts to be functionally silent. The OOF uAUG is created within an IF uORF, which hides its effect, as either element alone is sufficient to repress translation. Therefore, the variant sequence is not interpretable by Perturbation as the uAUGs are not found. A Scrambler trained with a low entropy penalty ($t_{bits} = 0.125$, $\lambda = 1$) detects both possible interpretations. With a higher penalty ($\lambda = 10$), the Scrambler must find a smaller set of salient features and marks only the OOF uAUG as important, which is sufficient to explain the repression. However, rather than just identifying

'ATG', the Scrambler identifies the subsequence 'ATGA', even though the trailing adenine is not important for the explanation. This occurs because the number of salient features is highly variable in any given 5' UTR, and so a Scrambler trained for a target entropy t_{bits} may 'over-interpret' some examples.

Per-example Fine-tuning Removes Interpreter Artifacts—To overcome issues with over-interpretation for datasets with a highly variable number of important features, we apply a per-example fine-tuning step: starting with $\mathcal{S}(\boldsymbol{x})$, we optimize new scores $\boldsymbol{s}$ by gradient descent which are specific to $\boldsymbol{x}$ and remove excessive features of $\mathcal{S}(\boldsymbol{x})$ by maximizing the entropy of the PSSM unbounded. Specifically, $\boldsymbol{s}$ is optimized to subtract scores from $\mathcal{S}(\boldsymbol{x})$ in order to maximize the PSSM entropy $\text{KL}[\tilde{\boldsymbol{b}} \| \hat{\boldsymbol{x}}_{\boldsymbol{s}}]$ unbounded, while still minimizing the predictive reconstruction error $\text{KL}[\mathcal{P}(\boldsymbol{x}_{\boldsymbol{s}}^{(k)}) \| \mathcal{P}(\boldsymbol{x})]$. As seen with the variant example (Fig. 4d), fine-tuning of the Low entropy-penalty Scrambler removes the trailing 'A'. We also compared the Scrambler with and without fine-tuning to interpret silent uORF-altering and gain-of-function mutations (Extended Data Fig. 4d). Importantly, when compared on one of the benchmarks of Fig. 4b, we find that applying per-example fine-tuning to the pre-trained Scrambler scores produces more robust attribution results than individual per-example optimization of importance scores (Extended Data Fig. 4e). This is likely because the latter approach can become stuck in local minima or produce poorly generalizable interpretations, essentially 'overfitting' to spurious predictor signals.

Dropout for Alternate Open Reading Frame Discovery—Sequences with multiple IF uORFs are examples of patterns with redundant salient feature sets. By adding an importance score dropout layer to the Inclusion-Scrambler and training it on random dropout patterns, we obtain a model capable of dynamically proposing different IF uORFs as attributions (Extended Data Fig. 4f-g). Without dropout and a low entropy penalty ($t_{\text{bits}} = 0.125$, $\lambda = 1$), the Scrambler marks all IF starts and stops (i.e. all IF uORFs) as important in the example shown in Extended Data Fig. 4f. However, with a higher penalty ($\lambda = 10$), the Scrambler finds a smaller interpretation with only one IF uORF. When using dropout patterns to exclude either of the IF starts and stops, the Scrambler dynamically finds the alternate operands.

The Determinants of Protein-Protein Interactions

Here we apply Scramblers to interpret predicted interactions for pairs of proteins. This can be challenging, as proteins are defined along a narrow manifold of stably folded sequences. Any interpretation method must ensure that masked or perturbed sequences stay in distribution for predictions to remain accurate. We focused on a set of rationally designed coiled-coil heterodimers, where binding specificity is induced by hydrogen bond networks (HBNets) at the dimer interface [59, 60]. Using approximately 180, 000 designed heterodimers as positive training data and randomly paired monomers as a negative set, we trained a recurrent neural network (RNN) to predict dimer binding (Fig. 5a, Extended Data Fig. 5a; AUC = 0.96 on held-out test data, $n = 26, 459$). These coiled-coil proteins follow a conserved heptad repeat structure, but binder sequences of different lengths have their heptad motifs shifted by different offsets. Consequently, when training Scrambler networks

on this data, we used a separate background distribution for each sequence length. A test set of 478 designed heterodimers was used to investigate how well different attribution methods could capture HBNet positions and residues important for complex stability.

Unlike the DNA attribution tasks above, each dimerization prediction involves two input sequences. Consequently, there are multiple ways in which to define the Scrambler architecture. We first tested a *joint* Occlusion-Scrambler (Fig. 5b; left), which receives both sequences as input and can learn to recognize partner-specific binding features. After training, this architecture identified a subset of HBNet positions and hydrophobic residues at the interface necessary for binding specifically to the cognate partner (Fig. 5d). We also tested a *siamese* architecture, which saw only one input sequence at a time and was therefore constrained to learn global, partner-independent features (Fig. 5b; right). This architecture learned to identify nearly all HBNets, as these are the best determinants of *binding specificity* to any possible partner (Fig. 5c-d). We compared these Scrambler architectures to other attribution methods based on how much the positions with largest absolute-valued score either preserved or perturbed the RNN predictions (Fig. 5e; $t_{\text{bits}} = 0.25$, Inclusion; $t_{\text{bits}} = 2.4$, Occlusion). The siamese Inclusion-Scrambler and joint Occlusion-Scrambler had the best (lowest and highest) median KL-divergence for these tasks.

Importance Scores Reflect Dimer Stability and Hydrogen Bonds—Next, using in-silico Alanine scanning, we estimated the mean difference in $\Delta\Delta G$ between the original and mutated dimers for the eight most important residues per binder and attribution method [61]. We also compared the methods by their ability to discover HBNet positions based on their importance scores. We found that the joint Occlusion-Scrambler identified residues that destabilized the complex the most (mean difference $\Delta\Delta G = 2.20$, Fig. 5f), while the siamese model discovered significantly more HBNet positions (AUPRC = 0.61). These results support the idea that the siamese architecture learned discriminative features for both interacting and non-interacting pairs – HBNet residues – while the joint model learned features important for positive interaction, such as hydrophobic residues at the interface. While Integrated Gradients and DeepSHAP perturbed the RNN predictions more when considering only their positive-valued scores, the Scramblers had better $\Delta\Delta G$ scores and HBNet discovery rates (Extended Data Fig. 5b). This suggests that using a parametric masking model results in more generalizable features by overcoming spurious RNN signals. We performed additional comparisons to other attribution methods in Extended Data Fig. 5c-d and 6. We found that the temperature-based masking operator of Eq. 1 consistently outperforms other masks, that model-based interpretations were better than per-example masking approaches, and that methods such as L2X and INVASE failed to converge when trained on the full set of binders.

Finally, we tested the Scrambler against different versions of the ‘hot-deck’ SIS method (Extended Data Fig. 5e) [39, 40]: For each iteration of SIS, we sampled a number of background sequences and used these to mask de-selected features. The mean sample prediction was used as the function value. Similarly, we varied the number of Gumbel patterns sampled from the Scrambler PSSM during training (K in Eq. 2). The Scrambler operated well with few (≥ 4) samples and consistently outperformed SIS with 32 samples, both in predictive reconstruction and ground-truth comparisons. Additionally, the total

number of predictor queries required to interpret the entire dataset with comparable quality was ~ 70 times lower than SIS. Interestingly, using a simple masking scheme such as mean features resulted in poor interpretations for the dimer RNN predictor.

Interpreting Protein Structure Predictions

Recent advances in deep learning have made homology-free protein structure prediction possible. Such neural networks use the primary sequence and corresponding multiple sequence alignment (MSA) as input to predict three-dimensional atom-atom distances and backbone torsion angles [11]. Here, we apply Scramblers to the predictor trRosetta [12] to detect important sequence features for predicted structures. Since it is computationally heavy to query the predictor – and would likely require many iterations to train a Scrambler network – we demonstrate the approach for a single protein at a time (Fig. 6a). A trainable vector of scrambler scores (PSSM temperatures) is used to perturb the input sequence and MSA to minimize or maximize the total KL-divergence between predicted contact distributions (see Methods for details). We tested our approach on one of the alpha-helical hairpin binders from the previous dimer data set (Fig. 6b). PSSMs for the hairpin binder were optimized for the Inclusion (Fig. 6c) and Occlusion objectives (Fig. 6d). Both PSSMs used hydrophobic leucines and a symmetry-breaking glycine in the hairpin region to reconstruct or distort the original hairpin structure prediction, which aligns well with previous results [60]. We further tested the method on naturally occurring proteins with more complex 3D structures (Extended Data Fig. 7a-b).

Next, we altered the Scrambling architecture to enable MSA-free interpretation of *de novo* engineered proteins lacking natural sequence homology (Extended Data Fig. 7c). We optimized Inclusion-PSSMs to reconstruct the structure prediction of four proteins designed by deep network hallucination [62] (Fig. 6e). Important features found by the Scrambler included hydrophobic residues in the core for all four proteins. For validation, each protein's structure prediction was relaxed, and the per-residue breakdown of the Rosetta score function was computed [63]. We found that the per-residue energy values agreed reasonably well with the Scrambler importance scores (mean top 10 residue agreement = 0.425). However, note that some features marked by the Scrambler as important for the structure prediction are missed by the Rosetta energy breakdown. For example, as can be seen in Fig. 6e, the Scrambler marked loop glycines between alpha-helices as important for the structure prediction, even though they do not have a large energetic contribution to overall stability. Glycines frequently occur in loops and turns and are thought to be important for maintaining protein loop structure due to their size and flexibility compared to other residues [64, 65]. The MSA-free approach was also tested on a hairpin-like protein engineered by Activation Maximization (Extended Data Fig. 7d-e) [66].

Discussion

Here we introduced a new attribution method – Scrambler Networks – that builds on and extends earlier work in input masking approaches. The goal is to explain an input pattern for a machine learning model by finding the smallest set of features which either reconstruct or destroy its prediction [32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. Scrambler networks are

based on learning a parametric model that predicts a set of real-valued importance scores given an input pattern. These predicted scores are used as sampling temperatures within a PSSM that interpolates between the current input and a non-informative background distribution. We demonstrated the performance of Scramblers on several attribution tasks, ranging from identifying cis-regulatory elements in RNA sequences to protein-protein interactions and protein tertiary structure. Furthermore, Scramblers can be trained to find not only reconstructive, but also enhancing and repressive features, and can be used to dynamically find multiple salient feature sets within a single input pattern. For prediction problems where the salient feature sets vary widely in size, per-example fine-tuning can be employed to correct positions that were misidentified as important.

At the core of mask-based interpretation is the masking operator, which must be carefully chosen according to the problem domain. Earlier work in computer vision masked input patterns by fading or blurring [32, 33, 34], and other methods for discrete variable selection masked features with zeros [35, 36]. Neither of these operators are suitable for predictors trained on biological sequences, which expect discrete one-hot encoded patterns as input and predict poorly on patterns outside of the training data manifold. The Scrambler masking operator is conceptually similar to ‘hot-deck’ masking proposed in the SIS method by Carter et al. [39] and in the interpretation method of Zintgraf et al. [38], where deactivated features are replaced with random samples from the marginal distribution of that input. Scramblers generalize this concept by maintaining a categorical feature sampling distribution at each position in the sequence, where the predicted temperature (inverse importance score) defines how random a sample from that position will be on a continuous scale. An immediate benefit is that, in contrast to SIS, Scramblers output real-valued importance scores that reflect an internal ranking of feature importance. This masking operator also shares similarities with the counterfactual masking introduced by Chang et al. [37], but the temperature-based operator is simpler as it does not require training a generative fill-in model.

We showed in extensive benchmarks that temperature-based masking outperforms the hot-deck masking done with SIS and any of the fade-, blur- or mean-masking schemes used in [32, 33, 34]. By training a parametric model, we end up with a more sample-efficient method at interpreting patterns than per-example methods [32, 39]. We also showed that Scramblers produced better interpretations than L2X [35] and INVASE [36], which either overfit their interpretations when trained on the reduced datasets or did not converge well when trained on the full data. This highlights the difficulty in jointly training a sparse feature selector and a predictor on very heterogeneous data and showcases the potential in applying post-hoc interpretation to rich, pre-trained predictors. By optimizing over a sufficiently large training set, Scramblers avoid overfitting to spurious per-example signals and learn to generalize feature importance. Generalizable attributions are highly desirable in genomics and proteomics as they allow us to infer higher-quality biological discoveries from trained predictors, thereby offering richer insights into molecular processes that were previously considered black boxes. We envision that Scramblers will be particularly useful for rational sequence design, where they can be employed to either validate known design rules (as demonstrated on the protein-protein interaction task) or – as demonstrated on the protein structure task – to discover determinants which data-driven design methods have produced.

Methods

Scrambling Neural Network Definition

Masking Operator—Let $\boldsymbol{x} \in \{0, 1\}^{N \times M}$ be a one-hot-encoded sequence and let $\mathcal{S}(\boldsymbol{x}) \in (0, \infty]^N$ be the corresponding real-valued importance scores predicted by Scrambler network $\mathcal{S}$. The channel-broadcasted importance scores $\dot{\mathcal{S}}(\boldsymbol{x}) \in (0, \infty]^{N \times M}$ are defined as $\dot{\mathcal{S}}(\boldsymbol{x})_{ij} = \dot{\mathcal{S}}(\boldsymbol{x})_i, \forall i, j$. $\dot{\mathcal{S}}(\boldsymbol{x})$ are used as (differentiable) interpolation coefficients in log-space between background distribution $\tilde{\boldsymbol{b}}$ and the current input pattern $\boldsymbol{x}$ according to Eq. 1 for the Inclusion objective. Conversely, the scrambling operation for the Occlusion-model is defined in Eq. 3 below. Here, the expression $\boldsymbol{x} \times \dot{\mathcal{S}}(\boldsymbol{x})$ has been replaced with $\boldsymbol{x} \,/\, \dot{\boldsymbol{S}}(\boldsymbol{x})$. Either of these formulas return a softmax-relaxed distribution (a Position-Specific Scoring Matrix; PSSM) $\hat{\boldsymbol{x}}_{\mathcal{S}} \in [0, 1]^{N \times M}$, which can be interpreted as a representation of the input sequence where the information content at position i has been perturbed according to the sampling temperature $\mathcal{S}(\boldsymbol{x})_i$ predicted by the Scrambler.

$$\hat{\boldsymbol{x}}_{\mathcal{S}} = \sigma\left(\log \tilde{\boldsymbol{b}} + \boldsymbol{x} \,/\, \dot{\mathcal{S}}(\boldsymbol{x})\right) \tag{3}$$

Given the scrambled PSSM $\hat{\boldsymbol{x}}_{\mathcal{S}}$, we sample approximate discrete one-hot-coded patterns $\boldsymbol{x}_{\mathcal{S}}^{(k)}$ using the Concrete distribution (or Gumbel-distribution) [44], enabling differentiation:

$$\boldsymbol{x}_{\mathcal{S}}^{(k)} = \{\mathbb{C}_i\}_{i=1}^{N}, \quad \mathbb{C}_i \sim \text{Concrete}(\hat{\boldsymbol{x}}_{\mathcal{S}i1}, \ldots, \hat{\boldsymbol{x}}_{\mathcal{S}iM}) \tag{4}$$

Alternatively, we sample exact discrete patterns $\boldsymbol{x}_{\mathcal{S}}^{(k)}$ from $\hat{\boldsymbol{x}}_{\mathcal{S}}$ and use the Softmax Straight-Through Estimator (ST-sampling) [43] to approximate the gradient $\nabla_{W_{\mathcal{S}}} \boldsymbol{x}_{\mathcal{S}}^{(k)}$ of the input pattern with respect to the Scrambler network weights. To reduce variance during training, we draw K samples $\{\boldsymbol{x}_{\mathcal{S}}^{(k)}\}_{k=1}^{K}$ at each iteration of gradient descent and walk down the average gradient (by default, $K = 32$).

Objective Functions—Given the scrambled PSSM $\hat{\boldsymbol{x}}_{\mathcal{S}}$, a collection of differentiable sequence samples $\{\boldsymbol{x}_{\mathcal{S}}^{(k)}\}_{k=1}^{K}$ and the background distribution $\tilde{\boldsymbol{b}}$, the Scrambler is trained by backpropagation to either minimize Eq. 2 (Inclusion) or Eq. 5 below (Occlusion). For predictor models $\mathcal{P}$ with a sigmoid or softmax output (classification), the prediction reconstruction error between $\mathcal{P}(\boldsymbol{x})$ and $\mathcal{P}(\boldsymbol{x}_{\mathcal{S}}^{(k)})$ is the standard cross-entropy error. For regression tasks, the cost is defined as the mean squared error. The cost parameter t_{bits} defines the target per-position PSSM conservation and λ defines the conservation penalty weight. With a smaller value of λ, the Scrambler is allowed to output PSSMs with a larger spread of conservation around the target t_{bits} (example: for one input pattern, the scrambled PSSM may have a mean conservation of 0.05, for another pattern, it may be 0.25). With a larger λ, the conservation values are forced closer to t_{bits}.

$$\min_{\mathcal{S}} -\left(\frac{1}{K}\sum_{k=1}^{K}\mathrm{KL}[\mathscr{P}(\boldsymbol{x}_{\mathcal{S}}^{(k)})\|\mathscr{P}(\boldsymbol{x})]\right)+\lambda\cdot\left(t_{\mathrm{bits}}-\frac{1}{N}\cdot\mathrm{KL}[\widetilde{\boldsymbol{b}}\|\hat{\boldsymbol{x}}_{\mathcal{S}}]\right)^2 \tag{5}$$

Rather than optimizing the Scrambler to reconstruct predictions using a small feature set, we can train the model to find feature sets that either positively or negatively influence the prediction. For classification tasks, we either maximize or minimize the log-probability $\log \mathscr{P}(\boldsymbol{x}_{\mathcal{S}}^{(k)})$ (Eq. 6-7). For regression tasks, we either maximize or minimize the regressor $\mathscr{P}(\boldsymbol{x}_{\mathcal{S}}^{(k)})$ (Eq. 8-9).

$$\min_{\mathcal{S}} -\left(\frac{1}{K}\sum_{k=1}^{K}\log\mathscr{P}(\hat{\boldsymbol{x}}_{\mathcal{S}}^{(k)})\right)+\lambda\cdot\left(t_{\mathrm{bits}}-\frac{1}{N}\cdot\mathrm{KL}[\widetilde{\boldsymbol{b}}\|\hat{\boldsymbol{x}}_{\mathcal{S}}]\right)^2 \tag{6}$$

$$\min_{\mathcal{S}} -\left(\frac{1}{K}\sum_{k=1}^{K}\log\left(1-\mathscr{P}(\hat{\boldsymbol{x}}_{\mathcal{S}}^{(k)})\right)\right)+\lambda\cdot\left(t_{\mathrm{bits}}-\frac{1}{N}\cdot\mathrm{KL}[\widetilde{\boldsymbol{b}}\|\hat{\boldsymbol{x}}_{\mathcal{S}}]\right)^2 \tag{7}$$

$$\min_{\mathcal{S}} -\left(\frac{1}{K}\sum_{k=1}^{K}\mathscr{P}(\hat{\boldsymbol{x}}_{\mathcal{S}}^{(k)})\right)+\lambda\cdot\left(t_{\mathrm{bits}}-\frac{1}{N}\cdot\mathrm{KL}[\widetilde{\boldsymbol{b}}\|\hat{\boldsymbol{x}}_{\mathcal{S}}]\right)^2 \tag{8}$$

$$\min_{\mathcal{S}}\left(\frac{1}{K}\sum_{k=1}^{K}\mathscr{P}(\hat{\boldsymbol{x}}_{\mathcal{S}}^{(k)})\right)+\lambda\cdot\left(t_{\mathrm{bits}}-\frac{1}{N}\cdot\mathrm{KL}[\widetilde{\boldsymbol{b}}\|\hat{\boldsymbol{x}}_{\mathcal{S}}]\right)^2 \tag{9}$$

Note that we could have defined the conservation penalty of Eq. 2-5 and 6-9 in many other ways. The KL-divergence $\mathrm{KL}[\widetilde{\boldsymbol{b}}\|\hat{\boldsymbol{x}}_{\mathcal{S}}]$ corresponds to the generalized entropy of $\hat{\boldsymbol{x}}_{\mathcal{S}}$ and is particularly suitable for the Inclusion-Scrambler: the expression reaches 0 when $\hat{\boldsymbol{x}}_{\mathcal{S}}$ is maximally entropic (with respect to $\widetilde{\boldsymbol{b}}$). For the Occlusion-Scrambler where entropy should be minimized, it may seem more appropriate to minimize the KL-divergence $\mathrm{KL}[\hat{\boldsymbol{x}}_{\mathcal{S}}\|\boldsymbol{x}]$ to the original input pattern rather than maximizing $\mathrm{KL}[\widetilde{\boldsymbol{b}}\|\hat{\boldsymbol{x}}_{\mathcal{S}}]$, as this cost corresponds to the cross-entropy of $\hat{\boldsymbol{x}}_{\mathcal{S}}$ with respect to $\boldsymbol{x}$ and is 0 at minimum entropy. Alternatively, for both Scramblers, we could simply minimize the area scrambled by the importance scores, $\sigma(\mathcal{S}(\boldsymbol{x}))\times 2-1$ (we refer to these as *lum* values). Overall, $\mathrm{KL}[\widetilde{\boldsymbol{b}}\|\hat{\boldsymbol{x}}_{\mathcal{S}}]$ resulted in better performance, which we hypothesize has to do with the information content of $\widetilde{\boldsymbol{b}}$ otherwise being lost.

Background Distribution—The background distribution $\widetilde{\boldsymbol{b}}$ is intended to keep the scrambled sequence samples in distribution and along the manifold of valid patterns, no matter where on the interpolation line between $\widetilde{\boldsymbol{b}}$ and $\boldsymbol{x}$ the scrambled distribution $\hat{\boldsymbol{x}}_{\mathcal{S}}$ is. There are several possible choices of $\widetilde{\boldsymbol{b}}$, with varying degrees of complexity:

$$(\text{0-BG})\ \widetilde{\boldsymbol{b}}_{ij} = P(x_i = j) = \frac{\sum_{k=1}^{N} \mathbf{1}_{x_i^{(k)} = j}}{N} \tag{10}$$

$$(\text{H-BG})\ \widetilde{\boldsymbol{b}}_{ij}(\boldsymbol{x}) = P(x_i = j \mid x_{i-H}, \ldots, x_{i+H}) = \frac{\sum_{k=1}^{N} \mathbf{1}_{x_i^{(k)} = j \cap x_{i-H}^{(k)} = \ldots}}{\sum_{k=1}^{N} \mathbf{1}_{x_{i-H}^{(k)} = \ldots}} \tag{11}$$

$$(\text{DATA})\ \widetilde{\boldsymbol{b}}_{ij} = x_{ij}^{(\mathbb{K})}, \quad \mathbb{K} \sim \{1, \ldots, N\} \tag{12}$$

$$(\text{GEN})\ \widetilde{\boldsymbol{b}}_{ij}(\boldsymbol{x}) = \mathcal{G}(\boldsymbol{z}, \boldsymbol{x}, \mathcal{S}(\boldsymbol{x}))_{ij}, \quad \boldsymbol{z} \sim \mathcal{N}(0, 1)^d \tag{13}$$

The simplest background distribution, 0-BG, corresponds to the mean input pattern across the training set (i.e. marginal feature probabilities). A more complex background, H-BG, is the distribution of a particular feature conditioned on its neighboring features [38]. Finally, the DATA- and GEN distributions consist of input patterns randomly chosen from the dataset (DATA), or patterns generated by a distribution-capturing model such as a conditional GAN or Variational Autoencoder (GEN) [37]. In all of our experiments, we relied on the simplest background distribution, 0-BG, since all tested sequence-predictive models behaved well on such samples. Still, for more narrowly defined manifolds, H-BG or DATA/GEN may be more appropriate to stay in distribution.

Network Architecture—The Scrambler is based on a Residual Neural Network architecture with dilated convolutions (Extended Data Fig. 1) [42]. For the MNIST, 5' UTR, and protein binder attribution tasks, the network consisted of 5 groups of 4 dilated residual blocks (20 blocks in total), with filter width 3. For the alternative polyadenylation task, the network had just one group of 4 residual blocks, with filter width 8.

Mask Dropout and Biasing—Scramblers can find multiple salient feature set solutions for an input pattern using input-masking layers such as dropout- or biasing layers, which either disallow or enforce specific sequence positions in the retained feature set. Specifically, we let $\boldsymbol{D}, \boldsymbol{T} \in \{0, 1\}^N$ be the dropout- and biasing masks and apply them to the importance scores by element-wise multiplication and addition respectively:

$$\hat{\boldsymbol{x}}_{\mathcal{S}} = \sigma(\log \widetilde{\boldsymbol{b}} + \boldsymbol{x} \times (\dot{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{D}, \boldsymbol{T}) \times \dot{\boldsymbol{D}} + \dot{\boldsymbol{T}} \cdot c)) \tag{14}$$

Here c is the temperature of the biasing mask $\boldsymbol{T}$ (smaller c allows more randomness in selecting the letter at a biased position, a larger c makes it more likely to select the current input letter at the biased position). The Scrambler network $\mathcal{S}$ also receives $\boldsymbol{D}$ and $\boldsymbol{T}$ as additional input, allowing the network to learn to output alternate scores conditioned on which positions were dropped or enforced. During training, we use random samples of $\boldsymbol{D}$ and $\boldsymbol{T}$ (various training schemes are illustrated in Extended Data Fig. 2; for the MNIST task,

we used all three schemes; for the 5' UTR task we used only the first scheme, i.e. uniform random training patterns consisting of squares). After training on randomized dropout and biasing patterns, we can provide hand-crafted patterns at inference time to detect alternate feature sets of new input examples.

Fine-tuning and Per-example Optimization—For heterogeneous datasets where the number of important features per example is highly variable, we apply a per-example fine-tuning step to the importance scores predicted by the (pre-trained) Scrambler network $\mathcal{S}$ to remove any excessive (or redundant) features. Specifically, for each input pattern $\boldsymbol{x}$, we predict Scrambler scores $\mathcal{S}(\boldsymbol{x})$ and optimize the *subtraction scores* $\boldsymbol{s}$ according to Eq. 15-16 (for Inclusion) or Eq. 17-18 (for Occlusion) below. The scrambling Eq. is reformulated with the expression $\max[\dot{\mathcal{S}}(\boldsymbol{x}) - \boldsymbol{s}, \boldsymbol{0}]$; this forces the resulting fine-tuned importance scores to select a subset of the features identified by the original scores $\mathcal{S}(\boldsymbol{x})$, i.e., the new scores cannot find a new alternate explanation for example $\boldsymbol{x}$. This restriction prevents 'overfitting' to the predictor, which is otherwise a problem with per-example optimization. Architecturally, we do not optimize $\boldsymbol{s}$ directly; we optimize parameters $\boldsymbol{w}$, which are instance-normalized and softplus-transformed ($\boldsymbol{s}$ = Softplus(IN($\boldsymbol{w}$))). In our experiments, we optimize $\boldsymbol{w}$ for 300 iterations of gradient descent (Adam, learning rate = 0:01).

$$\min_{\boldsymbol{s}} \left(\frac{1}{K} \sum_{k=1}^{K} \mathrm{KL}[\mathscr{P}(\boldsymbol{x}_{\boldsymbol{s}}^{(k)}) \| \mathscr{P}(\boldsymbol{x})] \right) + \lambda \cdot \frac{1}{N} \cdot \mathrm{KL}[\tilde{\boldsymbol{b}} \| \hat{\boldsymbol{x}}_{\boldsymbol{s}}] \tag{15}$$

$$\hat{\boldsymbol{x}}_{\boldsymbol{s}} = \sigma\left(\log \tilde{\boldsymbol{b}} + \boldsymbol{x} \times \max[\dot{\mathcal{S}}(\boldsymbol{x}) - \boldsymbol{s}, \boldsymbol{0}] \right) \tag{16}$$

$$\min_{\boldsymbol{s}} - \left(\frac{1}{K} \sum_{k=1}^{K} \mathrm{KL}[\mathscr{P}(\boldsymbol{x}_{\boldsymbol{s}}^{(k)}) \| \mathscr{P}(\boldsymbol{x})] \right) - \lambda \cdot \frac{1}{N} \cdot \mathrm{KL}[\tilde{\boldsymbol{b}} \| \hat{\boldsymbol{x}}_{\boldsymbol{s}}] \tag{17}$$

$$\hat{\boldsymbol{x}}_{\boldsymbol{s}} = \sigma\left(\log \tilde{\boldsymbol{b}} + \boldsymbol{x} \,/\, \max[\dot{\mathcal{S}}(\boldsymbol{x}) - \boldsymbol{s}, \boldsymbol{\epsilon}] \right), \lim_{\epsilon \to +0} \tag{18}$$

For comparison, we also perform per-example optimization from scratch (i.e., we start with randomly initialized values rather than pre-trained Scrambler scores). To this end, we optimize Eq. 15 (for Inclusion) or Eq. 17 (for Occlusion). The perturbation operators are identical to Eq. 1 and 3, but with individual scores $\boldsymbol{s}$ per pattern $\boldsymbol{x}$: $\hat{\boldsymbol{x}}_{\boldsymbol{s}} = \sigma(\log \tilde{\boldsymbol{b}} + \boldsymbol{x} \times \dot{\boldsymbol{s}})$ (Inclusion), $\hat{\boldsymbol{x}}_{\boldsymbol{s}} = \sigma(\log \tilde{\boldsymbol{b}} + \boldsymbol{x} \,/\, \dot{\boldsymbol{s}})$ (Occlusion).

Attribution Tasks and Predictors

MNIST: The predictor was a CNN with 2 convolutional layers, 1 max-pool layer and a single fully connected hidden layer. The network was trained for this study. Predicts a 10-way classification score $\mathscr{P}(\boldsymbol{x}) \in \mathbb{R}^{10}$ (softmax) given a binarized MNIST image $\boldsymbol{x} \in \{0, 1\}^{28 \times 28}$ as input.

APA (3' UTR): [7] The predictor was a pre-trained CNN (APARENT) which predicts relative alternative polyadenylation isoform abundance $\mathscr{P}(\boldsymbol{x}) \in [0, 1]$ given a 206 nt one-hot DNA sequence $\boldsymbol{x} \in \{0, 1\}^{206\times4}$ as input. The trained model was downloaded from: https://github.com/johli/aparent/tree/master/saved_models.

Translation (5' UTR): [10] The predictor was a CNN (Optimus 5-Prime) which predicts mean ribosome load $\mathscr{P}(\boldsymbol{x}) \in \mathbb{R}$ given a 50 nt one-hot DNA sequence $\boldsymbol{x} \in \{0, 1\}^{50\times4}$ as input. The model was re-trained for this study but the training procedure was based on the code from: https://github.com/pjsample/human_5utr_modeling.

Protein heterodimer binders: The predictor was a RNN with siamese recurrent GRU layers, a dropout layer and a single fully connected hidden layer. The network was trained for this study. Predicts protein dimer binding probability $\mathscr{P}(\boldsymbol{x}) \in [0, 1]$ (sigmoid activation) given two right-padded, 81 residues long protein binder sequences $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \{0, 1\}^{81\times20}$ as input. Note: When training Scrambler networks on these data, we used a separate background distribution $\tilde{\boldsymbol{b}}$ for each unique protein sequence length.

Protein structure: [12] Predicts distance and angle distributions of backbone atoms in 3D given an input primary sequence. The trained model was downloaded from: https://files.ipd.uw.edu/pub/trRosetta/model2019_07.tar.bz2.

Given a one-hot-encoded sequence pattern $\boldsymbol{x} \in \{0, 1\}^{N\times20}$ and Multiple Sequence Alignment $\mathbf{MSA} \in \{0, 1\}^{K\times N\times21}$, trRosetta predicts the atom-atom distance distribution $\boldsymbol{D}(\boldsymbol{x}, \mathbf{MSA}) \in [0, 1]^{N\times N\times37}$ and backbone torsion angle distributions $\boldsymbol{\theta}(\boldsymbol{x}, \mathbf{MSA})$, $\boldsymbol{\omega}(\boldsymbol{x}, \mathbf{MSA}) \in [0, 1]^{N\times N\times24}$, $\boldsymbol{\phi}(\boldsymbol{x}, \mathbf{MSA}) \in [0, 1]^{N\times N\times12}$. In the paper, we performed per-example scrambling to interpret example protein structures, i.e. we did not train a discriminative Scrambler model to predict importance scores. Instead, this method optimizes a set of importance scores s for each pattern $\boldsymbol{x}$ that we wish to interpret. In the context of inclusion-scrambling, we find the smallest set of residues that minimizes the total KL-divergence of the target structure:

$$\begin{aligned}\min_{\boldsymbol{s}} \frac{1}{K}\Bigg(\sum_{k=1}^{K} & \mathrm{KL}[\boldsymbol{D}(\boldsymbol{x}_{\boldsymbol{s}}^{(k)}, \mathbf{MSA}_{\boldsymbol{s}}^{(k)}) \| \boldsymbol{D}(\boldsymbol{x}, \mathbf{MSA})] \\ & + \mathrm{KL}[\boldsymbol{\theta}(\boldsymbol{x}_{\boldsymbol{s}}^{(k)}, \mathbf{MSA}_{\boldsymbol{s}}^{(k)}) \| \boldsymbol{\theta}(\boldsymbol{x}, \mathbf{MSA})] \\ & + \mathrm{KL}[\boldsymbol{\omega}(\boldsymbol{x}_{\boldsymbol{s}}^{(k)}, \mathbf{MSA}_{\boldsymbol{s}}^{(k)}) \| \boldsymbol{\omega}(\boldsymbol{x}, \mathbf{MSA})] \\ & + \mathrm{KL}[\boldsymbol{\phi}(\boldsymbol{x}_{\boldsymbol{s}}^{(k)}, \mathbf{MSA}_{\boldsymbol{s}}^{(k)}) \| \boldsymbol{\phi}(\boldsymbol{x}, \mathbf{MSA})] \Bigg) \\ & + \lambda \cdot \left(t_{\mathrm{bits}} - \frac{1}{N} \cdot \mathrm{KL}[\tilde{\boldsymbol{b}} \| \hat{\boldsymbol{x}}_{\boldsymbol{s}}] \right)^2\end{aligned}$$

For occlusion-scrambling, we optimize the inverse objective:

$$\min_{\boldsymbol{s}} \frac{1}{K}\Bigg(-\sum_{k=1}^{K} \mathrm{KL}[\boldsymbol{D}(\boldsymbol{x}_{\boldsymbol{S}}^{(k)}, \mathbf{MSA}_{\boldsymbol{S}}^{(k)}) \| \boldsymbol{D}(\boldsymbol{x}, \mathbf{MSA})] - \mathrm{KL}[\boldsymbol{\theta}(\boldsymbol{x}_{\boldsymbol{S}}^{(k)}, \mathbf{MSA}_{\boldsymbol{S}}^{(k)}) \| \boldsymbol{\theta}(\boldsymbol{x}, \mathbf{MSA})] - \mathrm{KL}[\boldsymbol{\omega}(\boldsymbol{x}_{\boldsymbol{S}}^{(k)}, \mathbf{MSA}_{\boldsymbol{S}}^{(k)}) \| \boldsymbol{\omega}(\boldsymbol{x}, \mathbf{MSA})] - \mathrm{KL}[\boldsymbol{\phi}(\boldsymbol{x}_{\boldsymbol{S}}^{(k)}, \mathbf{MSA}_{\boldsymbol{S}}^{(k)}) \| \boldsymbol{\phi}(\boldsymbol{x}, \mathbf{MSA})]\Bigg) + \lambda \cdot \left(t_{\mathrm{bits}} - \frac{1}{N} \cdot \mathrm{KL}[\tilde{\boldsymbol{b}} \| \hat{\boldsymbol{x}}_{\boldsymbol{S}}]\right)^2$$

In the above formulations, we define $\boldsymbol{x}_{\boldsymbol{S}}^{(k)}, \mathbf{MSA}_{\boldsymbol{S}}^{(k)} \sim \hat{\boldsymbol{x}}_{\boldsymbol{S}}, \widehat{\mathbf{MSA}}_{\boldsymbol{S}}$ (letter-by-letter Gumbel-sampling). For inclusion-scrambling, we define:

$$\hat{\boldsymbol{x}}_{\boldsymbol{S}} = \sigma(\log \tilde{\boldsymbol{b}} + \boldsymbol{x} \times \dot{\boldsymbol{s}})$$
$$\widehat{\mathbf{MSA}}_{\boldsymbol{S}} = \sigma(\log \tilde{\boldsymbol{b}} + \mathbf{MSA} \times \ddot{\boldsymbol{s}})$$

For occlusion-scrambling, we define:

$$\hat{\boldsymbol{x}}_{\boldsymbol{S}} = \sigma(\log \tilde{\boldsymbol{b}} + \boldsymbol{x} \,/\, \dot{\boldsymbol{s}})$$
$$\widehat{\mathbf{MSA}}_{\boldsymbol{S}} = \sigma(\log \tilde{\boldsymbol{b}} + \mathbf{MSA} \,/\, \ddot{\boldsymbol{s}})$$

Here $\dot{\boldsymbol{s}} \in \mathbb{R}^{N \times M}$ is a channel-broadcasted copy of the importance scores $\boldsymbol{s} \in \mathbb{R}^{N}$ and $\ddot{\boldsymbol{s}} \in \mathbb{R}^{K \times N \times M}$ is a MSA-broadcasted copy. We obtain $\boldsymbol{s}$ from a vector of real-valued (trainable) numbers that we instance-normalize and apply the softplus activation on. We train $\boldsymbol{s}$ until convergence using the Adam optimizer with learning rate = 0.01, $\beta_1 = 0.5$ and $\beta_2 = 0.9$.

Attribution Methods

The attribution methods listed below were included in all the benchmark comparisons. However, DeepLIFT- and Guided Backprop were excluded from the PPI task (Fig. 5) since they were incompatible with the heterodimer RNN predictor model. See the Supplemental Information for a list of additional attribution methods that were tested only for the PPI task.

Scrambler (Inclusion): Scrambler network trained to minimize Eq. 2 using the temperature-based perturbator of Eq. 1, with background $\tilde{\boldsymbol{b}}$ set to the mean input pattern across the training set. The network was trained between 20-50 epochs depending on task (when the validation error started to saturate).

Scrambler (Occlusion): Scrambler network trained to minimize Eq. 5 using the temperature-based perturbator of Eq. 3, with background $\tilde{\boldsymbol{b}}$ set to the mean input pattern across the training set.

Perturbation: Baseline attribution method where each nucleotide or residue in the input pattern $\boldsymbol{x}$ is systematically exchanged for every other possible letter, measuring the change in prediction as a proxy for the importance score. Let $\tilde{\boldsymbol{x}}^{(ij)}$ be the sequence

pattern corresponding to exchanging the current nucleotide/residue at position i for letter j. We then calculate the importance score s_i at position i as the absolute mean change in prediction across all letters j: $s_i = |1 / M \cdot \sum_{j=1}^{M} \mathscr{P}(\tilde{\boldsymbol{x}}^{(ij)}) - \mathscr{P}(\boldsymbol{x})|$. For classification tasks, we compare the log-scores of the predicted probabilities: $s_i = |1 / M \cdot \sum_{j=1}^{M} \log \mathscr{P}(\tilde{\boldsymbol{x}}^{(ij)}) - \log \mathscr{P}(\boldsymbol{x})|$.

Gradient Saliency [25]: The Gradient Saliency method was executed using the implementation from Ancona et al. [45]. We used either the absolute value of the saliency scores ('Gradient Saliency') or the signed scores ('Gradient Saliency (Signed)').

Code: https://github.com/marcoancona/DeepExplain.

Guided Backprop [27]: The Guided Backprop method was executed using the implementation from Shrikumar et al. [29]. We used either the absolute value of the saliency scores ('Guided Backprop') or the signed scores ('Guided Backprop (Signed)').

Code: https://github.com/kundajelab/deeplift.

DeepLIFT [29]: For the MNIST task, the RevealCancel method was used. For all other tasks (due to predictor compatibility issues), the Rescale implementation from Ancona et al. [45] was used instead. The reference was set to the mean sequence pattern (PSSM) of the data set. We used either the absolute value of the saliency scores ('DeepLIFT') or the signed scores ('DeepLIFT (Signed)').

Code: (RevealCancel) [29] https://github.com/kundajelab/deeplift.

(Rescale) [45] https://github.com/marcoancona/DeepExplain.

DeepSHAP [30]: SHAP DeepExplainer ('DeepSHAP') was executed for each input pattern with 100 reference patterns sampled uniformly from the data set. We used either the absolute value of the saliency scores ('DeepSHAP') or the signed scores ('DeepSHAP (Signed)').

Code: https://github.com/slundberg/shap.

Integrated Gradients [28]: Integrated Gradients was executed using the implementation from Ancona et al. [45]. 10 integration steps were used per pattern. The reference was set to the mean sequence pattern (PSSM) of the data set. We used either the absolute value of the saliency scores ('Integrated Gradient') or the signed scores ('Integrated Gradients (Signed)').

Code: https://github.com/marcoancona/DeepExplain.

TorchRay [33]: The extremal preservation/perturbation method from the TorchRay package was executed using either the perturbation operator where masked pixels/letters are zeroed ('TorchRay Fade') or blurred by a gaussian ('TorchRay Blur'). For the sequence-predictive tasks, we changed the code base to support 1-dimensional patterns. In all tasks, we set the blur operator σ to 3, the mask operator σ was set to 5 or 3 and the 'step' argument was set to 2. The 'area' was set to either 0.1 or 0.2.

Code: https://github.com/facebookresearch/TorchRay.

Saliency Model [34]: The Saliency Model-method was trained using a ResNet-50 network architecture. For the sequence-predictive tasks, we changed the code base to support 1-dimensional input patterns. The 'encoder'-part of the network was not pre-trained and the 'selector'-component was not used. In all tasks, the blur kernel size was set to either 9 or 5 and σ was set to 3. For every training iteration, blur was used 50% of the time as the perturbation operator and fade (zeroing features) otherwise.

Code: https://github.com/PiotrDabkowski/pytorch-saliency.

Sufficient Input Subsets [39]: Sufficient Input Subsets (SIS) was executed with the threshold set to the 50th percentile of the predicted scores ($\mathscr{P}(\boldsymbol{x})$) across the data set. For input patterns with non-masked predictions below this threshold, we used a dynamic threshold of 0.8 times the non-masked predicted score.

We tested multiple masking schemes: (1) Replacing masked residue positions with the mean letter value ('SIS (Mean)') and (2) Replacing masked residue positions with letters sampled from the marginal letter distribution, repeating the sampling process X times, and calculating the mean prediction across all X samples ('SIS (X Sample)').

Code: https://github.com/google-research/google-research/tree/master/sufficient_input_subsets.

Zero Scrambler: Optimizes the same high-level objective as the Scrambler, but this model uses a perturbation operator similar to that of L2X [35] and INVASE [36] where de-selected features are zeroed. Specifically, 'Zero Scrambler' optimizes $\min_{\mathscr{M}} \mathrm{KL}[\mathscr{P}(\boldsymbol{x} \times \mathscr{M}^{(\mathrm{G})}(\boldsymbol{x}))\|\mathscr{P}(\boldsymbol{x})] + \lambda \cdot \left(t_{\mathrm{area}} - 1 \,/\, N \cdot \sum_{i=1}^{N} \mathscr{M}^{(\mathrm{S})}(\boldsymbol{x})_i\right)^2$, where $\mathscr{M}$ is a binary 0/1 masking model. The continuous-valued mask $\mathscr{M}^{(\mathrm{S})}(\boldsymbol{x})$ and binarized mask $\mathscr{M}^{(\mathrm{G})}(\boldsymbol{x})$ are obtained by applying sigmoid activations and Gumbel sampling on the output vector of $\mathscr{M}$ respectively. Internally, the network $\mathscr{M}$ is identical to the Scrambler.

L2X [35]: The 'approximator'-model used in the L2X procedure always had the same architecture as the original predictor of each task. The 'explainer'-model was the default network found in the example code, consisting of multiple layers of convolutions with global pooling to capture high-level features. The size parameter k was set to $0.1 \cdot N$ or $0.2 \cdot N$ depending on the task, where N is the length of the sequence. The L2X models were trained until convergence (early stopping on validation data).

Code: https://github.com/Jianbo-Lab/L2X/.

INVASE [36]: The 'critic'- and 'baseline'-models used in the INVASE procedure were standard convolutional neural networks with 3-5 convolutional layers, interlaced with instance-normalization layers, and one fully-connected hidden layer. The 'actor'-model was a convolutional neural network with 3-5 instance-normalized convolutional layers. The λ parameter varied between 0.05 and 50 depending on the task. The INVASE models were trained until convergence.

Code: https://github.com/jsyoon0823/INVASE.

Scrambler (Inclusion, Siamese): The 'siamese' Inclusion-Scrambler architecture used in the PPI task (Fig. 5). Given pairs of input patterns $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, we optimize $\min_{\mathcal{S}}\left(1 \,/\, K \cdot \sum_{k=1}^{K} \mathrm{KL}[\mathscr{P}(\boldsymbol{x}_{\mathcal{S},1}^{(k)}, \boldsymbol{x}_{\mathcal{S},2}^{(k)}) \| \mathscr{P}(\boldsymbol{x}_1, \boldsymbol{x}_2)]\right) + \lambda \cdot \left(t_{\text{bits}} - 1 \,/\, N \cdot \mathrm{KL}[\widetilde{\boldsymbol{b}}_1 \| \hat{\boldsymbol{x}}_{\mathcal{S},1}]\right)^2 + \lambda, \cdot \left(t_{\text{bits}} - 1 \,/\, N \cdot \mathrm{KL}[\widetilde{\boldsymbol{b}}_2 \| \hat{\boldsymbol{x}}_{\boldsymbol{S},2}]\right)^2$ where $\boldsymbol{x}_{\mathcal{S},1}^{(k)}, \boldsymbol{x}_{\mathcal{S},2}^{(k)} \sim \hat{\boldsymbol{x}}_{\mathcal{S},1}, \hat{\boldsymbol{x}}_{\mathcal{S},2}$ (letter-by-letter Gumbel-sampling) and $\hat{\boldsymbol{x}}_{\mathcal{S},\#} = \sigma(\log \widetilde{\boldsymbol{b}}_{\#} + \boldsymbol{x}_{\#} \times \dot{\mathcal{S}}(\boldsymbol{x}_{\#}))\,(\# = 1 \text{ or } 2)$. $\mathcal{S}$ is a (siamese) Scrambler network.

Scrambler (Occlusion, Joint): The 'joint' Occlusion-Scrambler architecture used in the PPI task (Fig. 5). Given pairs of input patterns $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, we optimize $\min_{\mathcal{S}} -(1 \,/\, K \cdot \sum_{k=1}^{K} \mathrm{KL}[\mathscr{P}(\boldsymbol{x}_{\mathcal{S},1}^{(k)}, \boldsymbol{x}_{\mathcal{S},2}^{(k)}) \| \mathscr{P}(\boldsymbol{x}_1, \boldsymbol{x}_2)]) + \lambda \cdot \left(t_{\text{bits}} - 1 \,/\, N \cdot \mathrm{KL}[\widetilde{\boldsymbol{b}}_1 \| \hat{\boldsymbol{x}}_{\mathcal{S},1}]\right)^2 + \lambda, \cdot \left(t_{\text{bits}} - 1 \,/\, N \cdot \mathrm{KL}[\widetilde{\boldsymbol{b}}_2 \| \hat{\boldsymbol{x}}_{\mathcal{S},2}]\right)^2$ where $\boldsymbol{x}_{\mathcal{S},1}^{(k)}, \boldsymbol{x}_{\mathcal{S},2}^{(k)} \sim \hat{\boldsymbol{x}}_{\mathcal{S},1}, \hat{\boldsymbol{x}}_{\mathcal{S},2}$ (letter-by-letter Gumbel-sampling) and $\hat{\boldsymbol{x}}_{\mathcal{S},\#} = \sigma\left(\log \widetilde{\boldsymbol{b}}_{\#} + \boldsymbol{x}_{\#} \,/\, \dot{\mathcal{S}}(\boldsymbol{x}_1, \boldsymbol{x}_2)\right)(\# = 1 \text{ or } 2)$. $\mathcal{S}$ is a two-input Scrambler network.

Scrambler (Occlusion, Siamese): The 'siamese' Occlusion-Scrambler architecture used in the PPI task (Fig. 5). Given pairs of input patterns $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, we optimize $\min_{\mathcal{S}} -(1 \,/\, K \cdot \sum_{k=1}^{K} \mathrm{KL}[\mathscr{P}(\boldsymbol{x}_{\mathcal{S},1}^{(k)}, \boldsymbol{x}_{\mathcal{S},2}^{(k)}) \| \mathscr{P}(\boldsymbol{x}_1, \boldsymbol{x}_2)]) + \lambda \cdot \left(t_{\text{bits}} - 1 \,/\, N \cdot \mathrm{KL}[\widetilde{\boldsymbol{b}}_1 \| \hat{\boldsymbol{x}}_{\mathcal{S},1}]\right)^2 + \lambda, \cdot \left(t_{\text{bits}} - 1 \,/\, N \cdot \mathrm{KL}[\widetilde{\boldsymbol{b}}_2 \| \hat{\boldsymbol{x}}_{\mathcal{S},2}]\right)^2$ where $\boldsymbol{x}_{\mathcal{S},1}^{(k)}, \boldsymbol{x}_{\mathcal{S},2}^{(k)} \sim \hat{\boldsymbol{x}}_{\mathcal{S},1}, \hat{\boldsymbol{x}}_{\mathcal{S},2}$ (letter-by-letter Gumbel-sampling) and $\hat{\boldsymbol{x}}_{\mathcal{S},\#} = \sigma\left(\log \widetilde{\boldsymbol{b}}_{\#} + \boldsymbol{x}_{\#} \,/\, \dot{\mathcal{S}}(\boldsymbol{x}_{\#})\right)(\# = 1 \text{ or } 2)$. $\mathcal{S}$ is a (siamese) Scrambler network.

Benchmarking Details

Synthetic IF uORF 5' UTR Datasets—Synthetic IF uORF 5' UTR sequences were generated for feature attribution with varying numbers of IF uORFS by selecting sequences from the Optimus 5-Prime *egfp_unmod_1* dataset [10], which contained no upstream starts or stops and had MRL values that fell between the 5th and 10th percentile of the dataset. This set of sequences ($n = 537$) had the original sequence nucleotides replaced by "ATG" and "TAG" uniformly at random at possible IF positions as needed to create the number of IF uORFs for the dataset, keeping the rest of the UTR fixed. Four synthetic 5' UTR datasets of $n = 512$ sequences were generated: one IF uORF by inserting an "ATG" and followed by a "TAG", two IF uORFs by inserting an "ATG" followed by two "TAG", two IF uORFs by inserting two "ATG" followed by "TAG," and four IF uORFs by inserting two "ATG" followed by two "TAG".

Hydrogen Bond Network Detection—The Chen et al. [60] dimers were designed to have a minimum of 4 residues involved, contacting all four helices, with all heavy-atom donors and acceptors in the network satisfied. However, in later design steps for the hetero-dimers, these networks were slightly disrupted making complete recovery difficult using the original parameters. As such, slightly relaxed criteria from the original design specifications were used to recover as many HBNet residues as

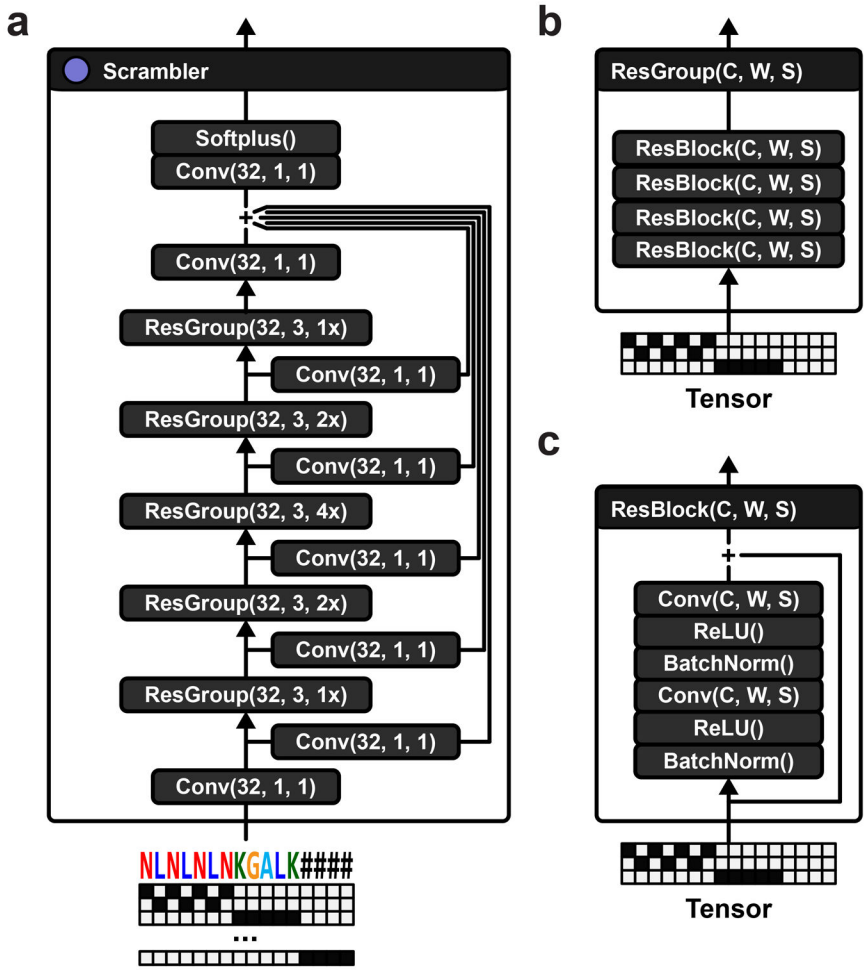
possible from the test set designed structures. HBNet residues were recovered from test coiled-coil dimers using PyRosetta [67] and the HBNetStapleInterface protocol, with the settings min_network_size=3, min_helices_contacted_by_network=3, hb_threshold=−0.3, and find_only_native_networks=True. The ref2015 score function was used during HBNet mover setup [63].

Mean AAG Difference Calculation and Alanine Scanning—Computational Alanine scanning was carried out for all residues in a hetero-dimer pair using PyRosetta [67]. Each position was mutated to Alanine and repacked with the neighboring residues prevented from design and repacking. The InterfaceAnalyzerMover was used with set_pack_input as False and set_pack_separated as True to calculate the mean mutation $\Delta\Delta G$ in Rosetta Energy Units (REU) at each position. For each attribution method, the eight most important residues per dimer were selected, and the difference of the mean $\Delta\Delta G$ of the top residues and the mean $\Delta\Delta G$ of all other residues was computed.

Per-Residue Score Function Breakdown and Substitution Scoring Matrix

Lysozyme and Sensor Histidine Kinase structures were predicted with trRosetta [12]. Each *de novo* structure was provided by [62]. All structures were relaxed using fast relax with PyRosetta [67], scored using the *beta_nov16* score function [63], and broken down into individual residue contributions. This was done ten times per structure, and the average residue contributions were calculated.

Extended Data



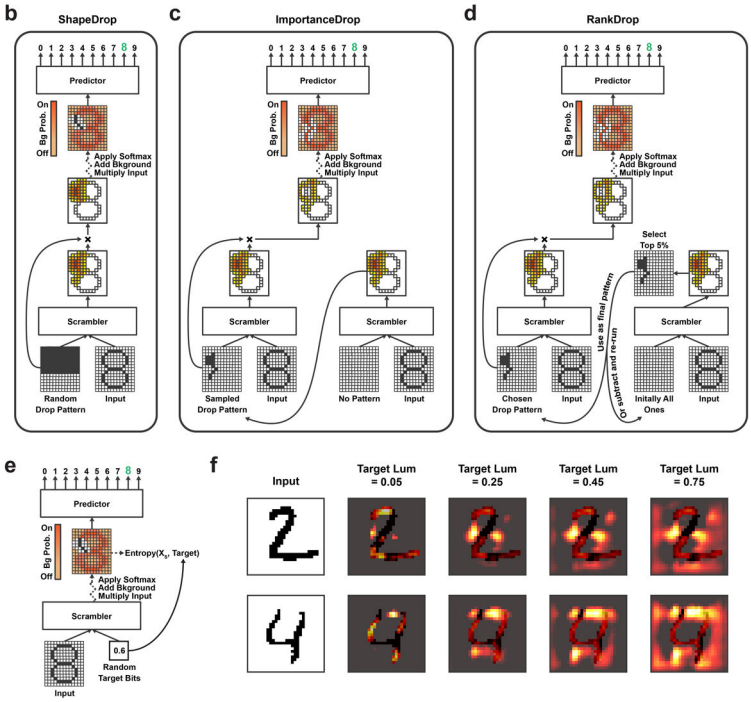
Extended Data Fig. 1.

(a) Scrambler network architecture. The network is based on groups of residual blocks. This particular network configuration has 5 groups of residual blocks, with 32 channels, filter width 3 and varying dilation factor (1x, 2x, 4x, 2x, 1x). There is a skip connection (single convolutional layer with 32 channels and filter width 1) before each residual group. All skip connections are added together with the output of the final residual group. A softplus activation is applied to the final tensor in order to get importance scores that are strictly larger than 0. (b) Each residual group consists of 4 identical residual blocks connected in series. (c) Each residual block consists of 2 dilated convolutions, each preceded by batch normalization and ReLU activations. A skip connection adds the input tensor to the output of the final convolution.

a

Median KL-Divs	Top % Features Included 20%	10%	5%	2%
SCRAMBLER	0.00	0.00	0.10	0.74
PERTURBATION	0.02	0.36	1.06	1.92
GRADIENT	0.00	0.17	0.88	2.04
GUIDED BACKPROP	0.00	0.06	0.72	1.91
DEEPLIFT	0.00	0.05	0.40	1.37
DEEPSHAP	0.00	0.02	0.30	1.25
INTEGRATED GRADIENTS	0.00	0.16	0.82	1.87
TORCHRAY BLUR	0.00	0.31	1.39	2.44
TORCHRAY FADE	0.00	0.29	1.56	2.41
SALIENCY MODEL	0.00	0.14	0.93	2.04
SIS MEAN	1.59	1.61	1.64	1.90
ZERO SCRAMBLER	0.41	1.30	2.10	2.68
L2X	0.27	1.05	2.04	2.71
INVASE	0.29	1.02	1.67	2.30

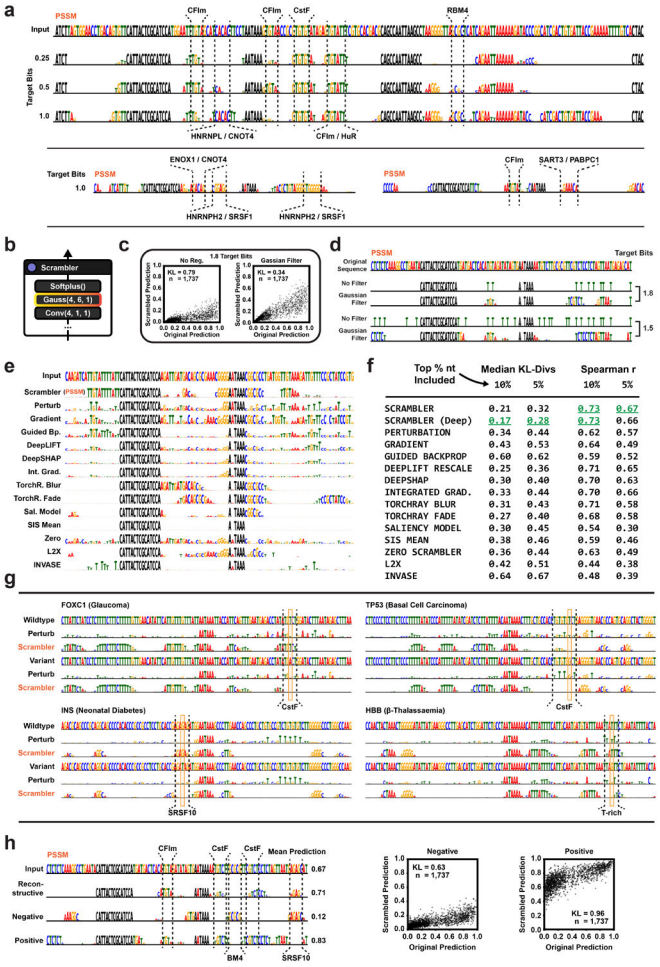
Mean Accuracies	Top % Features Included 100%	20%	10%	5%	2%
SCRAMBLER	0.99	0.99	0.99	0.87	0.01
PERTURBATION	0.99	0.79	0.43	0.09	0.00
GRADIENT	0.99	0.90	0.62	0.09	0.00
GUIDED BACKPROP	0.99	0.97	0.83	0.20	0.00
DEEPLIFT	0.99	0.93	0.82	0.35	0.00
DEEPSHAP	0.99	0.95	0.87	0.48	0.02
INTEGRATED GRADIENTS	0.99	0.94	0.66	0.14	0.00
TORCHRAY BLUR	0.99	0.92	0.48	0.04	0.00
TORCHRAY FADE	0.99	0.93	0.49	0.03	0.00
SALIENCY MODEL	0.99	0.90	0.66	0.12	0.00
SIS MEAN	0.99	0.00	0.00	0.00	0.00
ZERO SCRAMBLER	0.99	0.40	0.05	0.00	0.00
L2X	0.99	0.50	0.06	0.00	0.00
INVASE	0.99	0.49	0.07	0.00	0.00



Extended Data Fig. 2.

(a) Comparison of attribution methods on the 'Inclusion'-benchmark of Figure 2B (Perturbing the input patterns by keeping the top X% most important features according to each method and replacing all other features with random samples from a background distribution, n=1,888). Left table: Median KL-divergence between original predictions and predictions made on perturbed input patterns (lower is better). Right table: Classification accuracy of the predictor using the perturbed input patterns (higher is better). The '100%'-case refers to the original (non-perturbed) input pattern. The best method(s) are highlighted in green. (b) Uniformly random mask dropout training procedure, which teaches the Scrambler to find alternate salient feature sets. For each input pattern, we sample a random dropout pattern containing squares of varying width. The pattern is multiplied with the predicted importance scores, effectively zeroing out certain regions (forcing the background distribution to be used). The dropout pattern is also passed as additional input to the Scrambler (it is concatenated along the channel dimension), allowing the network to learn which other feature set to choose. (c) Biased dropout training procedure. Instead of randomly sampling dropout patterns, we first let the Scrambler predict importance scores with an all-ones dropout pattern (no dropout), which we use to form an importance sampling distribution. We then sample a dropout pattern and use it for training the same way we trained on uniformly random patterns. (d) Another biased dropout training approach. We

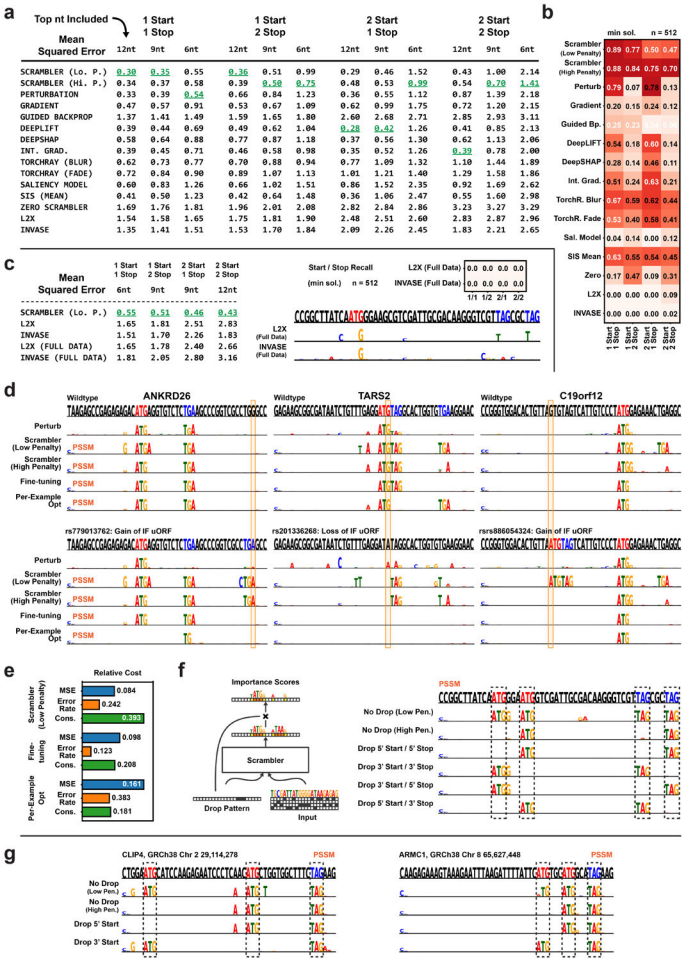
first use the Scrambler to predict the importance scores given the all-ones dropout pattern as input (no dropout). The top 5% most important features are subtracted from the all-ones pattern. Then, with a certain probability, we either re-run the Scrambler on this updated pattern (repeating the previous steps), or we end the loop and choose this as our final dropout pattern to train on. (e) Procedure for training the Scrambler to dynamically change the entropy of its solutions. Instead of fitting the network to a constant KL-divergence of its scrambled input distribution, we here randomly sample KL-divergence values and use them both as input to the network and as the target for the conservation penalty. The bit value is broadcasted and concatenated along the input channel dimension. (f) Example attributions of MNIST digit '2' and '4' with dynamically resized feature sets, by passing increasingly large target 'lum' values as input to the Scrambler ('lum' values are normalized KL-divergence bits, see Methods).



Extended Data Fig. 3.

(a) Top: Example attribution of a polyadenylation signal sequence from the APARENT test set, using Inclusion-Scramblers trained with increasingly large t_{bits} of conservation. Known regulatory motifs annotated. Bottom: Two additional example attributions, showing only the results for the t_{bits} = 1.0 case. (b) Non-trainable convolutional filter with a 1D gaussian kernel (filter width 6) is prepended to the final softplus activation function of the

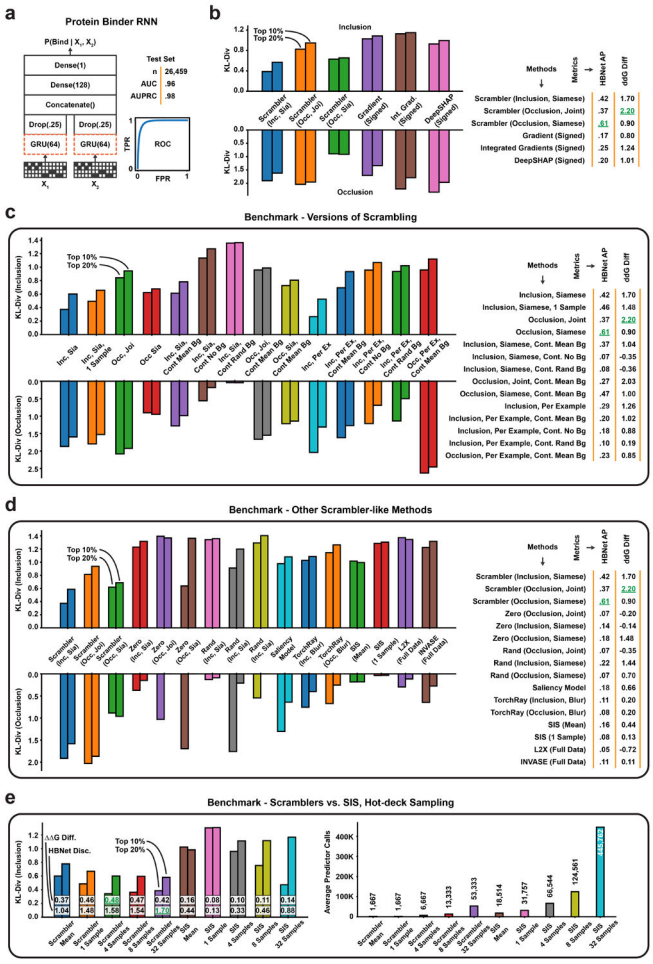
Scrambler. (c) APARENT isoform predictions of original sequences and of corresponding sampled sequences from the PSSMs predicted by an Occlusion-Scrambler trained with t_{bits} = 1.8, with and without the Gaussian filter. (d) Example attributions of t_{bits} = 1.8 - and 1.5 Occlusion-Scramblers, with and without the Gaussian filter. (e) Example attributions of a polyadenylation signal sequence, comparing different methods. (f) Comparison of attribution methods on the 'Inclusion'-benchmark of Figure 3B (Perturbing the input patterns by keeping the top X% most important features according to each method and replacing all other features with random samples from a background distribution, n=1,737). Median KL-divergences are computed between original predictions and predictions made on perturbed input patterns (lower is better). These predictions were made using the APARENT model. Shown are also the Spearman r correlation coefficients between original and perturbed predictions using the DeeReCT-APA model (higher is better). The default Scrambler network for the APA task uses 4 residual blocks, while the 'Deep' architecture uses a total of 20 residual blocks. The best method(s) are highlighted in green. (g) Attributions of four human polyadenylation signals which are associated with known deleterious variants, comparing the Perturbation method to the reconstructive Inclusion-Scrambler (t_{bits} = 0.25 target bits) on hypothetical variants which have not been found in the population. Gene names and clinical condition associated with the PAS annotated above each sequence. In each of the four examples, the Scrambler correctly detects the loss of the presumed RBP binding site or otherwise important motif due to each respective variant (loss of the CstF binding motif in FOXC1, TP53; loss of the SRSF10 binding motif in INS; loss of the T-rich DSE motif in HBB). (h) Left: Example attributions of a medium-strength polyadenylation signal sequence, using three Scramblers which have been optimized for different objectives: (Reconstructive features) reconstructing the original prediction, (Negative features) minimizing the prediction, and (Positive features) maximizing the prediction. Right: APARENT isoform predictions of original sequences and of corresponding sampled sequences from the PSSMs predicted by the Negative-feature and Positive-feature Scrambler respectively.



Extended Data Fig. 4.

(a) Benchmark comparison on the synthetic Start / Stop test sets, where input patterns are perturbed by keeping the most important features according to each method (6, 9 or 12 nt) and replacing all other features with random samples from a background distribution, n=512). Mean squared errors are computed between original predictions and predictions made on perturbed input patterns using the Optimus 5-Prime model (lower is better). We trained two Scramblers, one with a low entropy penalty (t_{bits} = 0.125, λ= 1) and one with a higher penalty (λ = 10). The best method(s) are highlighted in green. (b) Average recall for finding one of the start codons and one of the stop codons in the 6 most important nucleotides, as identified by each method, measured across the synthetic test sets. (c) Additional benchmark comparison for L2X and INVASE, when using the full 260,000 5' UTR dataset for training the interpreter model. Shown are the mean squared errors between predictions of original and perturbed input patterns, average recall for finding start and stop codons, and example visualizations on the synthetic start / stop test sets. (d) Left: Attribution of a ClinVar variant, rs779013762, in the ANKRD26 5' UTR, which is predicted by Optimus 5-Prime to be a functionally silent mutation. The variant creates an IF uORF overlapping an existing IF uORF. The per-example fine-tuning step (which starts from the Low entropy penalty-Scrambler scores) finds a minimal salient feature set in the variant sequence (one IF uORF), while the per-example optimization (which starts from randomly initialized scores)

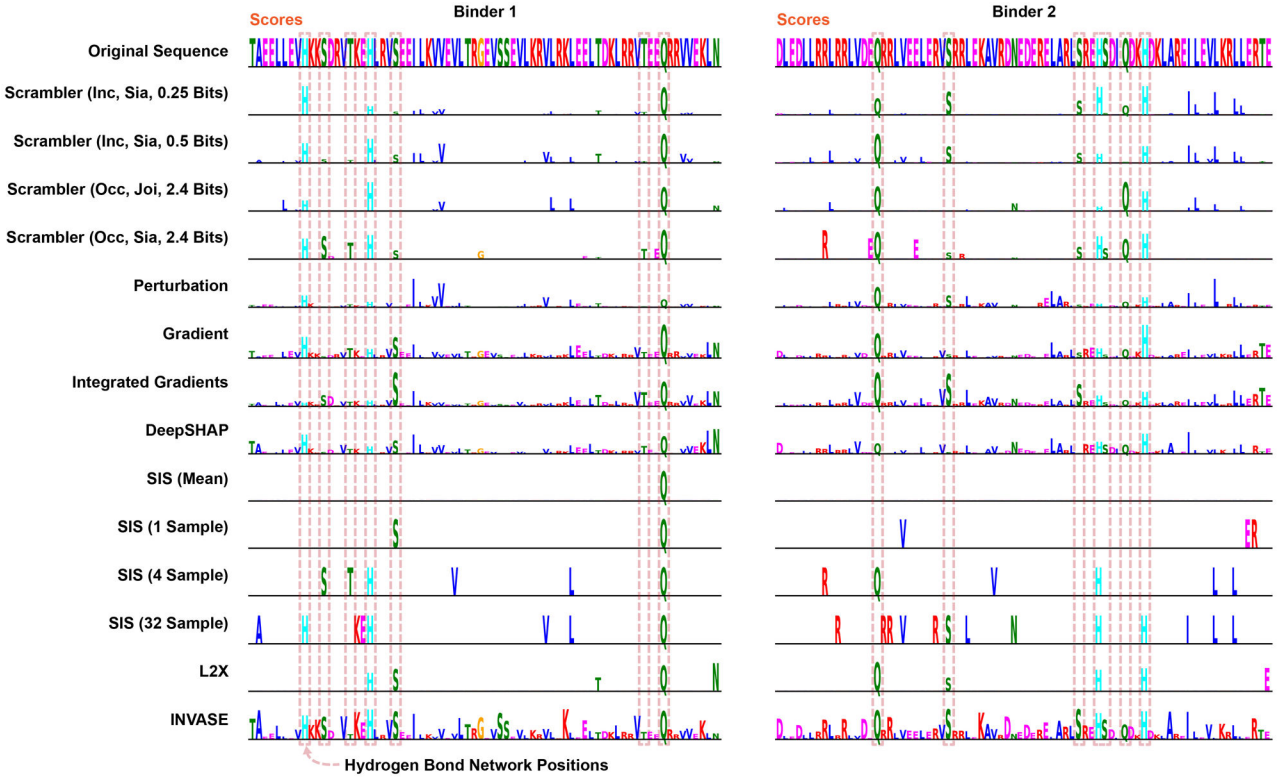
gets stuck in a local minimum. Middle: Attribution of a ClinVar variant, rs201336268, in the TARS2 5' UTR, which destroys two overlapping IF uORFs and is predicted to lead to upregulation. Both the fine-tuning step and the independent per-example optimization finds that no features are important in the variant sequence (both IF uORFs were removed by the variant and a fully random sequence has on average the same predicted MRL as the variant sequence). The Perturbation method has trouble explaining either of these variants due to saturation effects of the multiple IF stop codons. Right: Attribution of a rare variant, rs886054324, in the C19orf12 5' UTR, which creates two IF uORFs overlapping a strong OOF uAUG (hence a silent mutation). All attribution methods identify the OOF uAUG as the major determinant, however the Low entropy penalty-Scrambler incorrectly marks an (unmatched) stop codon in the wildtype sequence as important. Both the High entropy penalty-Scrambler and the fine-tuning step based off the Low penalty-Scrambler correctly filters the stop codon. (e) Benchmarking results on the 1 Start / 2 Stop dataset, comparing the Low entropy penalty-Scrambler network to running per-example fine-tuning of those scores and to the baseline method of optimizing each example from randomly initialized scores. Reported are the mean squared error between predictions on original and scrambled sequences ('MSE'), the error rate (1 - Accuracy) of not finding one Start codon and one Stop codon in the top 6 nt ('Error Rate'), and the mean per-nucleotide KL-divergence between the scrambled PSSM and the background PSSM ('Conservation'). (f) Example attributions using a Scrambler network trained with the mask dropout procedure (see Methods for details). By dropping different parts of the importance score mask, the Scrambler learns to discover alternate salient feature sets. In the example on the right: Finding alternate IF uORF regions by separately dropping each of the Start and Stop codons. (g) Example Scrambler attributions with the mask dropout mechanism on two native human 5' UTRs.



Extended Data Fig. 5.

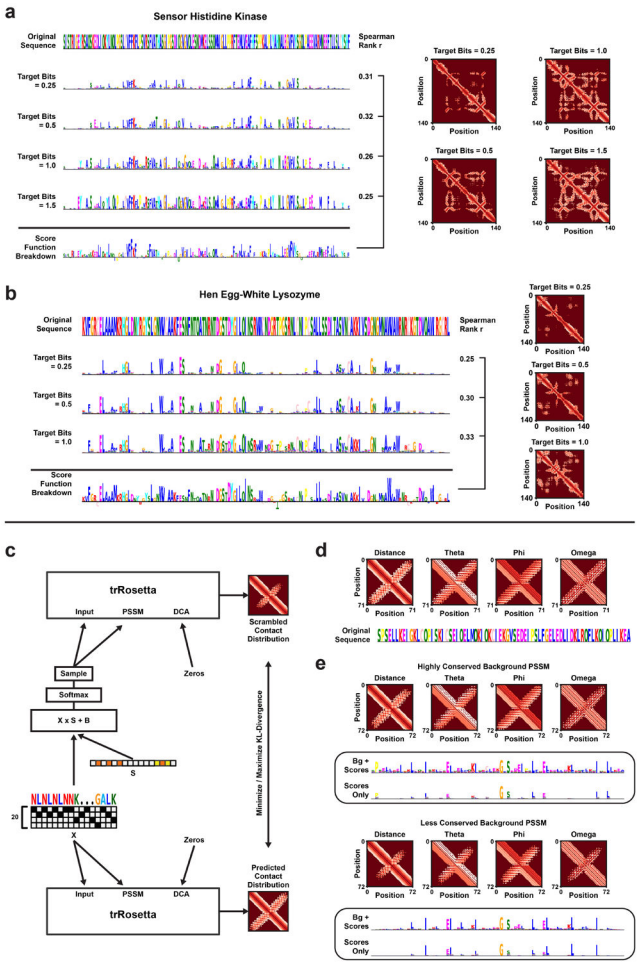
(a) Protein heterodimer binder RNN predictor, which was trained on computationally designed (dimerizing) pairs for positive data and randomly paired binders as negative data (see Methods for details). The RNN consists of a shared GRU layer, a dropout layer, and two fully-connected layers applied to the concatenated GRU output vectors. The final output (sigmoid activation) is treated as the Bind / No Bind classification probability. (b) Supplemental benchmark of Gradient Saliency, Integrated Gradients and DeepSHAP, using only the positive-valued importance scores. Left: Prediction KL-divergence of scrambled sequences compared to original test set sequences when either replacing all but the top X% most important amino acid residues with random samples (inclusion) or, conversely, when replacing the top X% nucleotides with random samples and keeping the remaining sequence fixed (occlusion). Right: Mean ddG Difference for the top 8 most important residues according to each method, measured across the test set, and HBNet Average Precision based on each method's importance scores. (c) Supplemental comparison of different versions of the Scrambling Neural Network (see Methods for a full description of each version). Left: KL-divergence benchmark based on the predictor RNN. Right: Mean ddG Differences and HBNet Discovery Precisions. (d) Supplemental comparison of other methods that optimize similar objectives as the Scrambler (see Methods for a full description of each method). Left: KL-divergence benchmark based on the predictor RNN. Right: Mean ddG Differences

and HBNet Discovery Precisions. (e) Supplemental comparison between Scrambling Neural Networks and Sufficient Input Subsets (SIS) with 'hot-deck' sampled masking (the number of samples used at each iteration is varied from 1 to 32; see Methods for details). Left: KL-divergence benchmark based on the predictor RNN, Mean ddG Differences and HBNet Discovery Precisions annotated on top of the bar chart. Right: Average number of predictor queries used to interpret a single input pattern (for the Scrambler, this is the amortized cost of training divided by the number of test patterns interpreted).



Extended Data Fig. 6.

Example attributions of a designed heterodimer binder pair, for a selection of benchmarked methods.



Extended Data Fig. 7.

(a) Four different Inclusion-PSSMs optimized to reconstruct the structural trRosetta prediction of a Sensor Histidine Kinase. Each PSSM is optimized for increasingly larger t_{bits}. The bottom sequence logo represents the Rosetta score function breakdown per residue (−REU). Spearman r ranged between 0.25 and 0.32 when comparing the absolute numbers of Rosetta energy values to the optimized importance scores. Shown is also the average structure prediction for 512 samples. (b) Inclusion-Scrambled PSSMs of the Hen Egg-white Lysozyme. The PSSM was re-optimized for three different target conservation bits. Spearman r ranged between 0.25 and 0.33 compared to the Rosetta score function. (c) Architecture for per-example scrambling of a single protein sequence according to the contact distributions predicted by trRosetta. Here, we do not use a Multiple Sequence Alignment (MSA), but instead pass the Gumbel-sampled sequence to the PSSM input and an all-zeros matrix to the DCA input. Total KL-divergence between trRosetta-predicted distributions (distance and angle-grams) of the original sequence and samples drawn from the scrambled PSSM is either minimized or maximized (inclusion or occlusion respectively). (d) Reference sequence and predicted contact distribution for a hairpin protein engineered by Activation Maximization. (e) Top: Inclusion-PSSM of the engineered hairpin protein, obtained after optimization with a highly conserved background distribution based on the

MSA. Bottom: Inclusion-PSSM of the engineered hairpin protein with a less conserved background distribution (smoothed with pseudo counts).

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

This work was supported by NIH award R21HG010945 and NSF award 2021552 to G.S. and NSF award 1908003 and 1703403 to S.K.

Data Availability

The data analyzed in this study originated from several previous publications and data repositories:

For the MNIST prediction task (Fig. 2), the data is available at (https://keras.io/api/datasets/mnist/). For the polyadenylation prediction task (Fig. 3), the data from Bogard et al. (2019) [7] is available on Gene Expression Omnibus (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE113849). For the 5' UTR prediction task (Fig. 4), the data from Sample et al. (2019) [10] is available on Gene Expression Omnibus (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE114002). For the protein-protein interaction prediction task (Fig. 5), the full data from Chen et al. (2019) [60] is available from the authors, upon request. The subset of sequences used in the benchmark comparisons are included in the Github repository (http://www.github.com/johli/scrambler). For the protein structure prediction task (Fig. 6), the *de novo* designed sequences from Anishchenko et al. (2020) [62] are available from the authors upon request.

References

[1]. Alipanahi B, Delong A, Weirauch M & Frey B Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nature Biotechnology 33, 831–838 (2015). URL 10.1038/nbt.3300.

[2]. Avsec Z et al. Base-resolution models of transcription-factor binding reveal soft motif syntax. Nature Genetics 53, 354–366 (2021). URL 10.1038/s41588-021-00782-6. [PubMed: 33603233]

[3]. Eraslan G, Avsec Z, Gagneur J & Theis F Deep learning: new computational modelling techniques for genomics. Nature Reviews Genetics 20, 389–403 (2019). URL 10.1038/s41576-019-0122-6.

[4]. Movva R et al. Deciphering regulatory DNA sequences and noncoding genetic variants using neural network models of massively parallel reporter assays. PLoS One 14, e0218073 (2019). URL 10.1371/journal.pone.0218073. [PubMed: 31206543]

[5]. Zhou J & Troyanskaya O Predicting effects of noncoding variants with deep learning-based sequence model. Nature Methods 12, 931–934 (2015). URL 10.1038/nmeth.3547. [PubMed: 26301843]

[6]. Arefeen A, Xiao X & Jiang T DeepPASTA: deep neural network based polyadenylation site analysis. Bioinformatics 35, 4577–4585 (2019). URL 10.1093/bioinformatics/btz283. [PubMed: 31081512]

[7]. Bogard N, Linder J, Rosenberg A & Seelig G A deep neural network for predicting and engineering alternative polyadenylation. Cell 178, 91–106 (2019). URL 10.1016/j.cell.2019.04.046. [PubMed: 31178116]

[8]. Cheng J et al. MMSplice: modular modeling improves the predictions of genetic variant effects on splicing. Genome Biology 20, 48 (2019). URL 10.1186/s13059-019-1653-z. [PubMed: 30823901]

[9]. Jaganathan K et al. Predicting splicing from primary sequence with deep learning. Cell 176, 535–548 (2019). URL 10.1016/j.cell.2018.12.015. [PubMed: 30661751]

[10]. Sample P et al. Human 5' UTR design and variant effect prediction from a massively parallel translation assay. Nature Biotechnology 37, 803–809 (2019). URL 10.1038/s41587-019-0164-5.

[11]. Senior A et al. Improved protein structure prediction using potentials from deep learning. Nature 577, 706–710 (2020). URL 10.1038/s41586-019-1923-7. [PubMed: 31942072]

[12]. Yang J et al. Improved protein structure prediction using predicted interresidue orientations. PNAS 117, 1496–1503 (2020). URL 10.1073/pnas.1914677117. [PubMed: 31896580]

[13]. Talukder A, Barham C, Li X & Hu H Interpretation of deep learning in genomics and epigenomics. Briefings in Bioinformatics (2020). URL 10.1093/bib/bbaa177.

[14]. Lanchantin J, Singh R, Lin Z & Qi Y Deep motif: Visualizing genomic sequence classifications. arXiv:1605.01133 (2016). URL https://arxiv.org/abs/1605.01133.

[15]. Schreiber J, Lu Y & Noble W Ledidi: Designing genome edits that induce functional activity. bioRxiv (2020). URL 10.1101/2020.05.21.109686.

[16]. Norn C et al. Protein sequence design by conformational landscape optimization. Proceedings of the National Academy of Sciences 118 (2021). URL 10.1073/pnas.2017228118.

[17]. Kelley D, Snoek J & Rinn J Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. Genome Res 26, 990–999 (2016). URL 10.1101/gr.200535.115. [PubMed: 27197224]

[18]. Zeng W, Wu M & Jiang R Prediction of enhancer-promoter interactions via natural language processing. BMC Genomics 19, 13–22 (2018). URL 10.1186/s12864-018-4459-6. [PubMed: 29298672]

[19]. Kelley D et al. Sequential regulatory activity prediction across chromosomes with convolutional neural networks. Genome Res 28, 739–750 (2018). URL 10.1101/gr.227819.117. [PubMed: 29588361]

[20]. Zeng W, Wang Y & Jiang R Integrating distal and proximal information to predict gene expression via a densely connected convolutional neural network. Bioinformatics 36, 496–503 (2020). URL 10.1093/bioinformatics/btz562. [PubMed: 31318408]

[21]. Singh S, Yang Y, Pczos B & Ma J Predicting enhancer-promoter interaction from genomic sequence with deep neural networks. Quant Biol 7, 122–137 (2019). URL 10.1007/s40484-019-0154-0. [PubMed: 34113473]

[22]. Calvo S, Pagliarini D & Mootha V Upstream open reading frames cause widespread reduction of protein expression and are polymorphic among humans. PNAS 106, 7507–7512 (2009). URL 10.1073/pnas.0810916106. [PubMed: 19372376]

[23]. Araujo P et al. Before it gets started: Regulating translation at the 5' UTR. Comp Funct Genomics 475731 (2012). URL 10.1155/2012/475731. [PubMed: 22693426]

[24]. Whiffin N et al. Characterising the loss-of-function impact of 5' untranslated region variants in 15,708 individuals. Nature Communications 11, 2523 (2020). URL 10.1038/s41467-019-10717-9.

[25]. Simonyan K, Vedaldi A & Zisserman A Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv:1312.6034 (2013). URL https://arxiv.org/abs/1312.6034.

[26]. Zeiler M & Fergus R Visualizing and understanding convolutional networks. In Computer Vision ECCV 2014 (818–833). URL 10.1007/978-3-319-10590-1_53.

[27]. Springenberg J, Dosovitskiy A, Brox T & Riedmiller M Striving for simplicity: The all convolutional net. arXiv:1412.6806 (2014). URL https://arxiv.org/abs/1412.6806.

[28]. Sundararajan M, Taly A & Yan Q Axiomatic attribution for deep networks. arXiv:1703.01365 (2017). URL https://arxiv.org/abs/1703.01365.

[29]. Shrikumar A, Greenside P & Kundaje A Learning important features through propagating activation differences. arXiv:1704.02685 (2017). URL https://arxiv.org/abs/1704.02685.

[30]. Lundberg S & Lee S-I A unified approach to interpreting model predictions. In Proceedings of the 31st international conference on neural information processing systems 4768–4777 (2017).

[31]. Singh M, Ribeiro S & Guestrin C Why should i trust you? explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining 1135–1144 (2018).

[32]. Fong R & Vedaldi A Interpretable explanations of black boxes by meaningful perturbation. In 2017 IEEE International Conference on Computer Vision ICCV 3449–3457 (2017). URL 10.1109/ICCV.2017.371.

[33]. Fong R, Patrick M & Vedaldi A Understanding deep networks via extremal perturbations and smooth masks. In 2019 IEEE/CVF International Conference on Computer Vision ICCV 2950–2958 (2019). URL 10.1109/ICCV.2019.00304.

[34]. Dabkowski P & Gal Y Real time image saliency for black box classifiers. arXiv:1705.07857 (2017). URL https://arxiv.org/abs/1705.07857.

[35]. Chen J, Song L, Wainwright M & Jordan M Learning to explain: An information-theoretic perspective on model interpretation. arXiv:1802.07814 (2018). URL https://arxiv.org/abs/1802.07814.

[36]. Yoon J, Jordon J & van der Schaar M Invase: Instance-wise variable selection using neural networks. In International Conference on Learning Representations (2018).

[37]. Chang C, Creager E, Goldenberg A & Duvenaud D Explaining image classifiers by counterfactual generation. arXiv:1807.08024 (2018). URL https://arxiv.org/abs/1807.08024.

[38]. Zintgraf L, Cohen T, Adel T & Welling M Visualizing deep neural network decisions: Prediction difference analysis. arXiv:1702.04595 (2017). URL https://arxiv.org/abs/1702.04595.

[39]. Carter B, Mueller J, Jain S & Gifford D What made you do this? understanding black-box decisions with sufficient input subsets. In The 22nd International Conference on Artificial Intelligence and Statistics 567–576 (2019).

[40]. Carter B et al. Critiquing protein family classification models using sufficient input subsets. J Comput Biol 27, 1219–1231 (2020). URL 10.1089/cmb.2019.0339. [PubMed: 31874057]

[41]. Covert I, Lundberg S & Lee S-I Feature removal is a unifying principle for model explanation methods. arXiv:2011.03623 (2020). URL https://arxiv.org/abs/2011.03623.

[42]. He K, Zhang X, Ren S & Sun J Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition CVPR 770–778 (2016). URL 10.1109/CVPR.2016.90.

[43]. Chung J, Ahn S & Bengio Y Hierarchical multiscale recurrent neural networks. arXiv:1609.01704 (2016). URL https://arxiv.org/abs/1609.01704.

[44]. Jang E, Gu S & Poole B Categorical reparameterization with gumbel-softmax. arXiv:1611.01144 (2016). URL https://arxiv.org/abs/1611.0114.

[45]. Ancona M, Ceolini E, ztireli C & Gross M Towards better understanding of gradient-based attribution methods for deep neural networks. arXiv:1711.06104 (2018). URL https://arxiv.org/abs/1711.06104.

[46]. Lecun Y, Bottou L, Bengio Y & Haffner P Gradient-based learning applied to document recognition. In Proceedings of the IEEE 86 2278–2324 (1998). URL 10.1109/5.726791.

[47]. Giammartino DD, Nishida K & Manley J Mechanisms and consequences of alternative polyadenylation. Mol Cell 43, 853–866 (2011). URL 10.1016/j.molcel.2011.08.017. [PubMed: 21925375]

[48]. Shi Y Alternative polyadenylation: new insights from global analyses. RNA 18, 2105–2117 (2012). URL 10.1261/rna.035899.112. [PubMed: 23097429]

[49]. Elkon R, Ugalde A & Agami R Alternative cleavage and polyadenylation: extent, regulation and function. Nature Reviews Genetics 14, 496–506 (2013). URL 10.1038/nrg3482.

[50]. Tian B & Manley J Alternative polyadenylation of mRNA precursors. Nature Reviews Molecular Cell Biology 18, 18–30 (2017). URL 10.1038/nrm.2016.116. [PubMed: 27677860]

[51]. Li Z et al. Deerect-apa: Prediction of alternative polyadenylation site usage through deep learning. Genomics, Proteomics and Bioinformatics (2021). URL 10.1016/j.gpb.2020.05.004.

[52]. Wylenzek M, Geisen C, Stapenhorst L, Wielckens K & Klingler K A novel point mutation in the 3' region of the prothrombin gene at position 20221 in a lebanese/syrian family. Thromb Haemost 85, 943–944 (2001). URL 10.1055/s-0037-1615777. [PubMed: 11372696]

[53]. Danckwardt S et al. The prothrombin 3'end formation signal reveals a unique architecture that is sensitive to thrombophilic gain-of-function mutations. Blood 104, 428–435 (2004). URL 10.1182/blood-2003-08-2894. [PubMed: 15059842]

[54]. Takagaki Y & Manley J RNA recognition by the human polyadenylation factor CstF. Mol Cell Biol 17, 3907–3914 (1997). URL 10.1128/MCB.17.7.3907. [PubMed: 9199325]

[55]. Stacey S et al. A germline variant in the TP53 polyadenylation signal confers cancer susceptibility. Nature Genetics 43, 1098–1103 (2011). URL 10.1038/ng.926. [PubMed: 21946351]

[56]. Medina-Trillo C et al. Rare foxc1 variants in congenital glaucoma: identification of translation regulatory sequences. Eur J Hum Genet 24, 672–680 (2016). URL 10.1038/ejhg.2015.169. [PubMed: 26220699]

[57]. Altay C et al. A mild thalassemia major resulting from a compound heterozygosity for the ivs-11-1 (g→a) mutation and the rare t→c mutation at the polyadenylation site. Hemoglobin 15, 327–330 (1991). URL 10.3109/03630269109027887. [PubMed: 1787101]

[58]. Garin I et al. Recessive mutations in the ins gene result in neonatal diabetes through reduced insulin biosynthesis. PNAS 107, 3105–3110 (2010). URL 10.1073/pnas.0910533107. [PubMed: 20133622]

[59]. Maguire J, Boyken S, Baker D & Kuhlman B Rapid sampling of hydrogen bond networks for computational protein design. J Chem Theory Comput 14, 2751–2760 (2018). URL 10.1021/acs.jctc.8b00033. [PubMed: 29652499]

[60]. Chen Z et al. Programmable design of orthogonal protein heterodimers. Nature 565, 106–111 (2019). URL 10.1038/s41586-018-0802-y. [PubMed: 30568301]

[61]. Ford A, Weitzner B & Bahl C Integration of the rosetta suite with the python software stack via reproducible packaging and core programming interfaces for distributed simulation. Protein Sci 29, 43–51 (2020). URL 10.1002/pro.3721. [PubMed: 31495995]

[62]. Anishchenko I, Chidyausiku T, Ovchinnikov S, Pellock S & Baker D De novo protein design by deep network hallucination. bioRxiv (2020). URL 10.1101/2020.07.22.211482.

[63]. Alford R et al. The rosetta all-atom energy function for macromolecular modeling and design. J Chem Theory Comput 13, 3031–3048 (2017). URL 10.1021/acs.jctc.7b00125. [PubMed: 28430426]

[64]. Parrini C et al. Glycine residues appear to be evolutionarily conserved for their ability to inhibit aggregation. Structure 13, 1143–1151 (2005). URL 10.1016/j.str.2005.04.022. [PubMed: 16084386]

[65]. Krieger F, Mglich A & Kiefhaber T Effect of proline and glycine residues on dynamics and barriers of loop formation in polypeptide chains. J Am Chem Soc 127, 3346–3352 (2005). URL 10.1021/ja042798i. [PubMed: 15755151]

[66]. Linder J & Seelig G Fast activation maximization for molecular sequence design. BMC Bioinformatics 22, 1–20 (2021). URL 10.1186/s12859-021-04437-5. [PubMed: 33388027]

[67]. Chaudhury S, Lyskov S & Gray J PyRosetta: a script-based interface for implementing molecular modeling algorithms using rosetta. Bioinformatics 26, 689–691 (2010). URL 10.1093/bioinformatics/btq007. [PubMed: 20061306]

[68]. Linder J et al. Code for: Interpreting Neural Networks for Biological Sequences by Learning Stochastic Masks. (Github, 2021). URL 10.5281/zenodo.5676173.

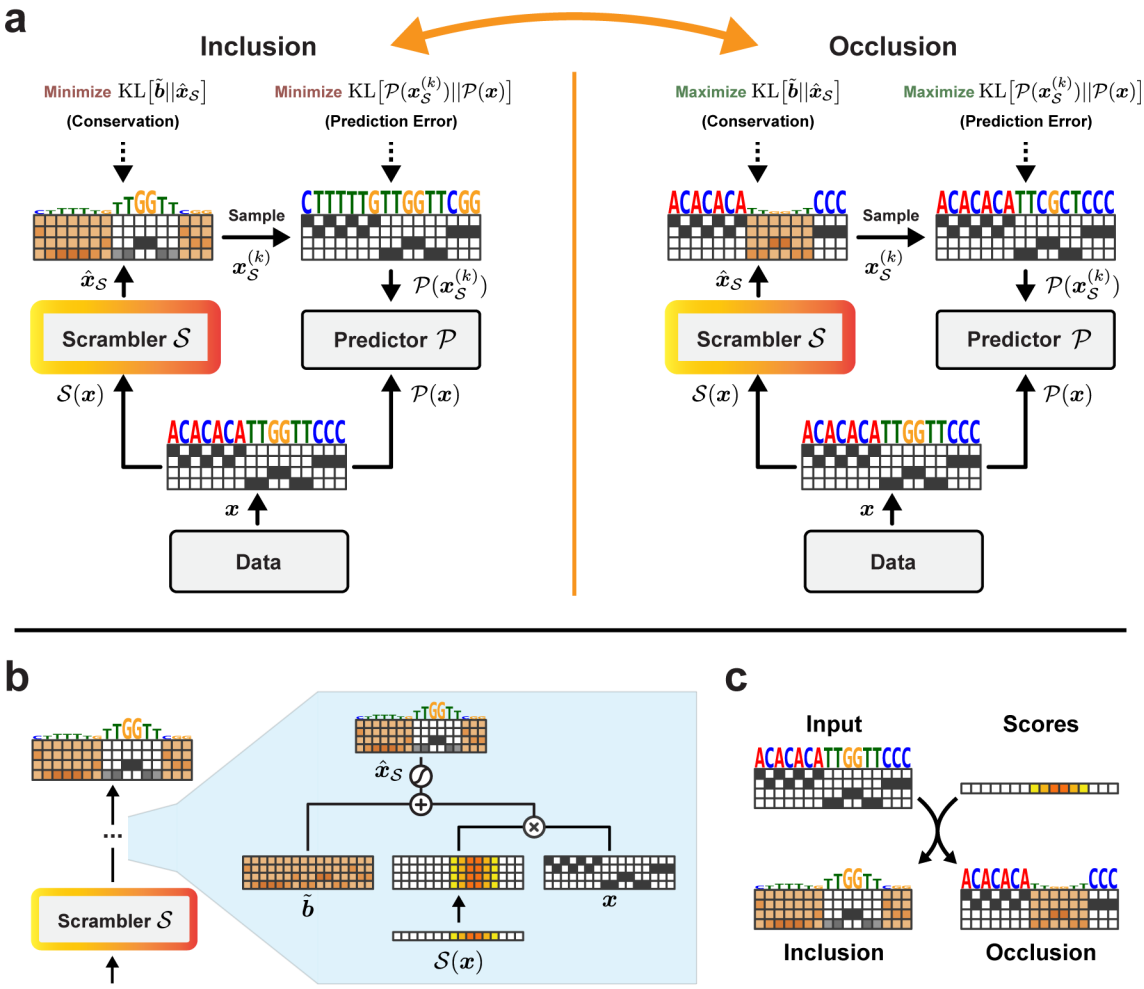


Figure 1:

Scrambler Architecture and Masking Operator (a) High-level architecture. Inclusion: Maximize the entropy of the PSSM predicted by the Scrambler and minimize the prediction error of samples drawn from it. Occlusion: Minimize PSSM entropy and maximize sample prediction error. (b) The temperature-based masking operation. The Scrambler predicts sampling temperatures ('importance scores') for the PSSM. (c) The Inclusion and Occlusion scrambling operations. Individual nucleotides (or amino acid residues) are perturbed by raising the sampling temperature of its corresponding categorical feature distribution.

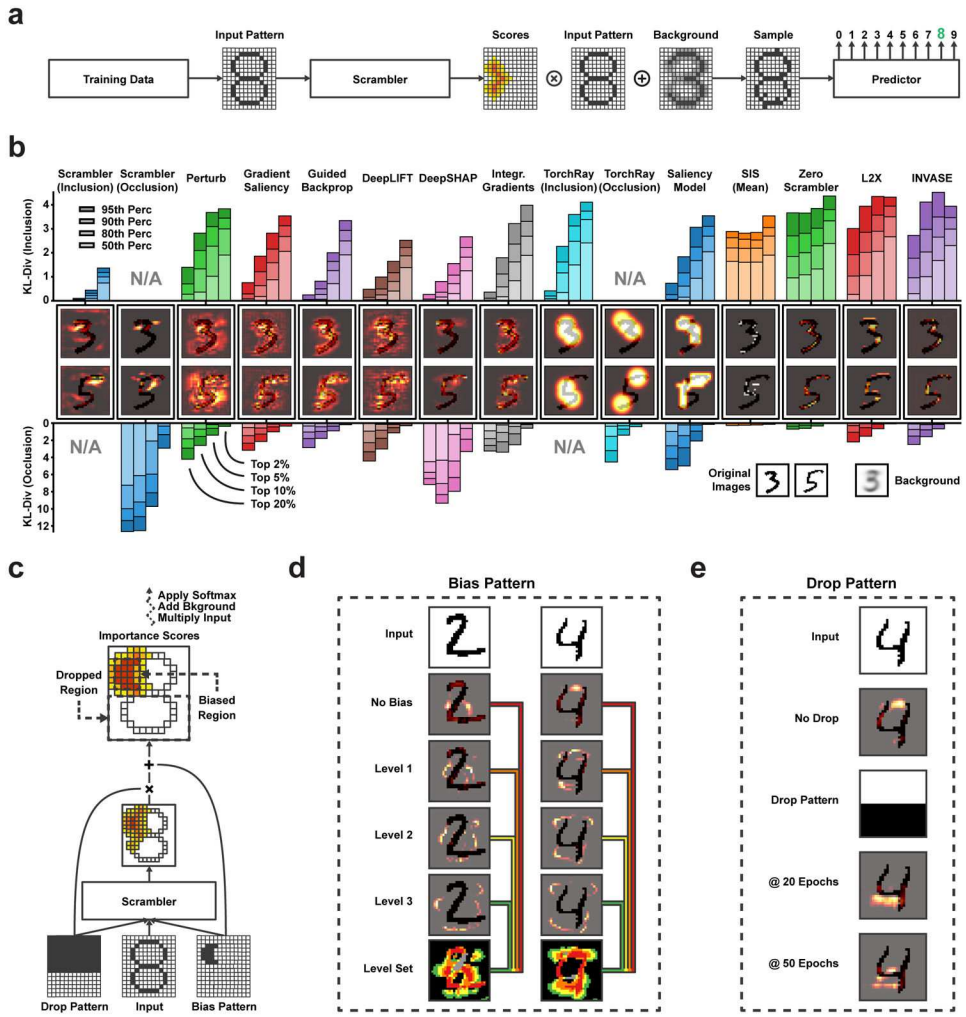


Figure 2:

MNIST Feature Attribution (a) Workflow for finding salient pixel regions within MNIST digits. The Scrambler learns to predict the salient feature set of the input image, which is superimposed on an uninformative background distribution. (b) Attributing feature importance for binarized MNIST digits '3' and '5'. (Upper bar chart) Keeping the top X% pixels according to importance scores and replacing all other pixels with random values. (Lower bar chart) Replacing the top X% pixels with random values. $n = 1,888$. (c) Importance score dropout and bias layers for finding alternate interpretations. (d) Example of dynamically sized feature sets using progressive bias patterns for differentiating digits '2' vs. '4'. The pixels found at each level are added to the next level of bias pattern to discover the next set of features. (e) Example of finding alternate salient features when dropping the upper half of the image importance scores.

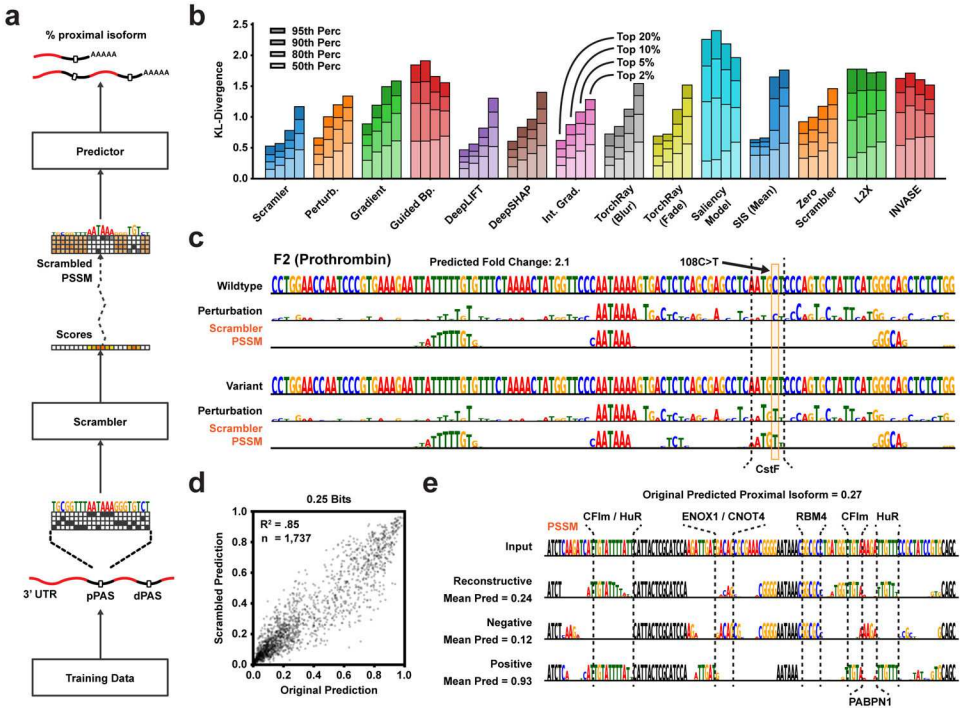


Figure 3:

APA Feature Attribution (a) Scrambler architecture for APA isoform attribution using the pre-trained model APARENT as the predictor. (b) Perturbing sequences by keeping the top *X*% nucleotides according to importance scores and replacing all other positions with random letters. The bar chart measures KL-divergence of APARENT-predictions between original- and perturbed test sequences ($n = 1, 737$). (c) Example attributions of a deleterious variant (*108C>T*) in the 3' UTR of the Prothrombin (*F2*) gene, comparing the Perturbation method to an Inclusion-Scrambler trained on the APARENT data. (d) APARENT predictions of original test set sequences compared to predictions made on sequence samples produced by the Scrambler PSSMs. (e) Example of Inclusion-Scramblers trained to reconstruct, maximise or minimize predictions, thus finding overall important, enhancing, or repressing motifs, respectively.

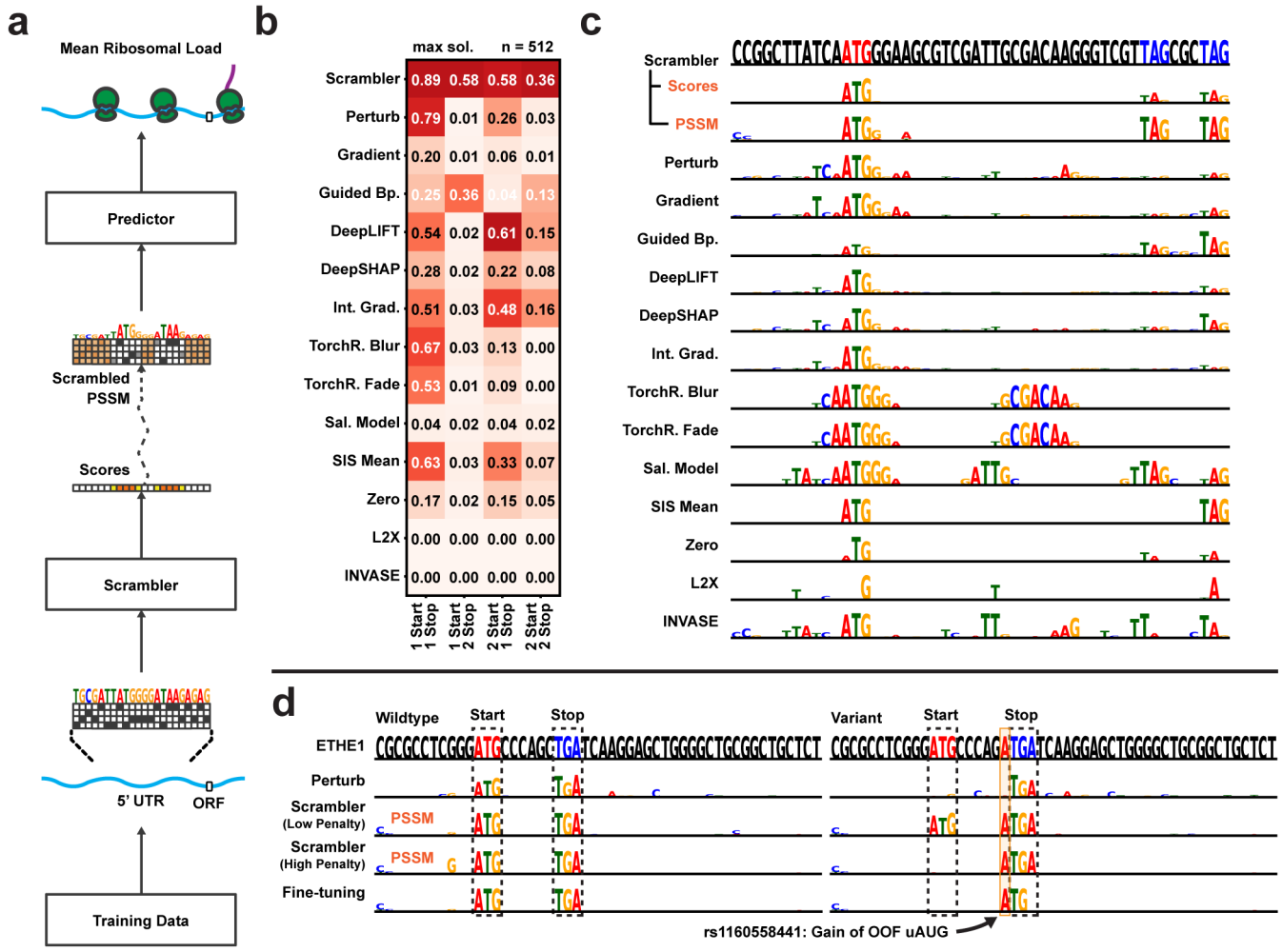


Figure 4:

5' UTR Feature Attribution (a) Scrambler architecture for 5' UTR ribosome load attributions using the pre-trained model Optimus 5' as the predictor. (b) Average recall of in-frame (IF) start and stop positions in a test set of synthetic IF uORF 5' UTRs. In this test, all start and stop codons must be among the highest scored nucleotides of a sequence to count as successfully recalled for a given method. $n = 512$. (c) Example attribution in a 5' UTR with multiple IF stop codons. (Top sequence) Red letters = IF start codon, Blue letters = IF stop codon. (d) Interpreting a rare, functionally silent mutation (*rs1160558441*) in the 5' UTR of the *ETHE1* gene, where an out-of-frame (OOF) upstream start codon (uAUG) is created within an in-frame (IF) upstream open reading frame (uORF). The 'Fine-tuning' optimization was performed on the importance scores predicted by the 'Low Penalty'-Scrambler.

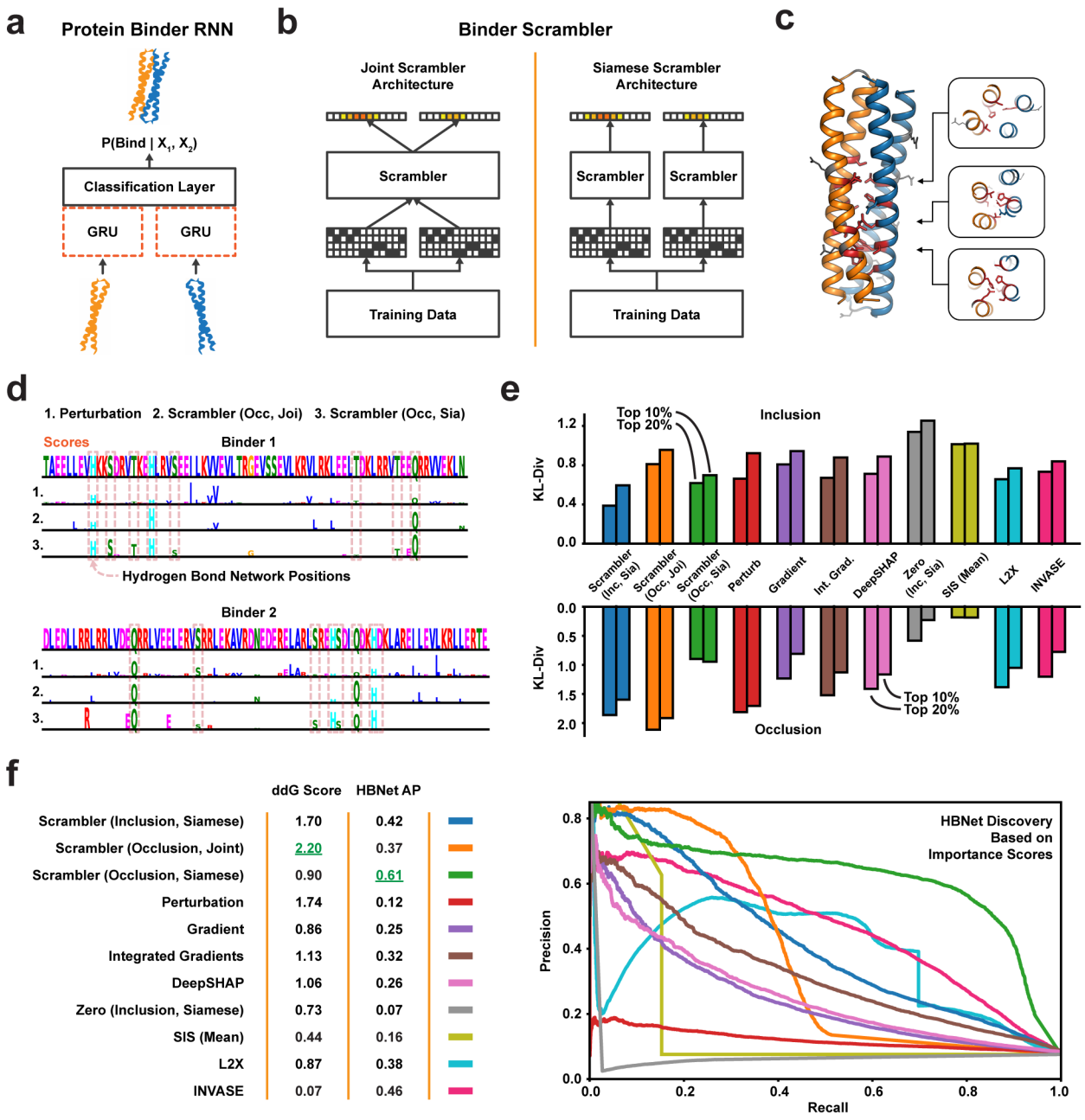


Figure 5:

Protein Heterodimer Feature Attribution (a) The protein binding predictor is a RNN based on a siamese GRU layer (orange dashed lines). (b) The *joint* and *siamese* Scrambler architectures for protein binder attribution. (c) 3D-visualization of an example binder pair structure. Discovered HBNet residues (by a siamese Occlusion-Scrambler) are shown in red licorice, important residues not part of the HB-net are shown as gray sticks. A cross-section of each of the three buried hydrogen bond networks is shown on the right. (d) Example attribution. Hydrogen bonds at the designed binding interface are marked with dashed red boxes. (e) Keeping the top X% residues according to the importance scores of each method and replacing the rest with random amino acids (top), or replacing the top X% with random

amino acids (bottom), measuring prediction KL-divergence. Zero refers to a Scrambler with a zero-based masking operator. (f) Left: *In silico* Alanine scanning benchmark. The 8 most important residues of each method were replaced with Alanines, measuring $\Delta\Delta G$ in Rosetta Energy Units (REU). Shown are the mean $\Delta\Delta G$ differences in a permutation test of 10,000 re-labelings. Right: Precision-recall curves and Average Precision (AP) for discovering the HBNet positions, based on the importance scores of each benchmarked method.

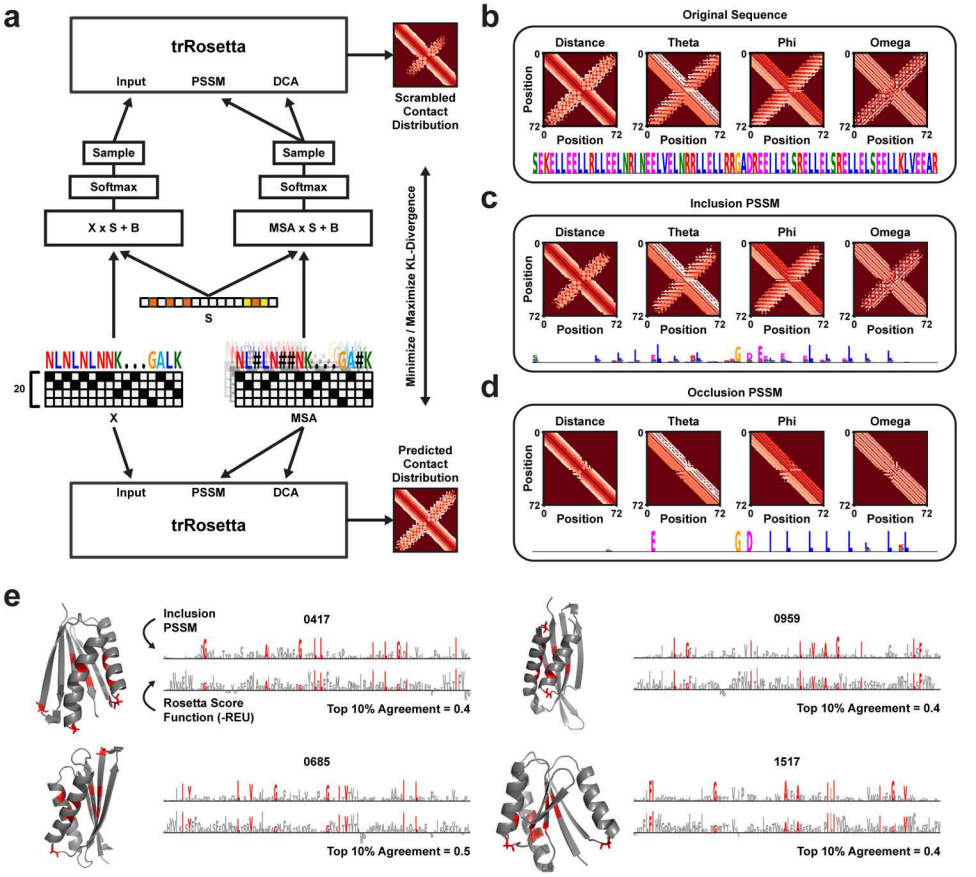


Figure 6:

Protein Structure Feature Attribution (a) The single-sequence Scrambling architecture for tr-Rosetta. Both the primary sequence ('X') and the multiple sequence alignment ('MSA') are scrambled with the same trainable mask ('S'). (b) Alpha-helical hairpin protein sequence and original structure prediction. (c) Inclusion-Scrambled PSSM of the hairpin protein, with the average predicted distance- and angle distributions for 512 samples drawn from the PSSM. (d) Occlusion-Scrambled PSSM of the hairpin protein and average structure prediction for 512 samples. (e) Inclusion-Scrambled PSSMs of four *de novo* proteins which have been designed by deep network hallucination. Each PSSM was optimized for t_{bits} = 1.0 target bits of conservation, using the amino acid frequency of naturally occurring sequences from the Protein Data Bank as background distribution. The bottom sequence logos represent the Rosetta score function breakdown per residue (−REU). The 10 most important residues according to the Scrambler scores are colored red.