Paratope Prediction using Convolutional and Recurrent Neural Networks

Edgar Liberis*, Petar Veličković*, Pietro Sormanni†, Michele Vendruscolo†, Pietro Liò*

***Abstract*— Antibodies play an essential role in the immune system of vertebrates and are vital tools in research and diagnostics. While hypervariable regions of antibodies, which are responsible for binding, can be readily identified from their amino acid sequence, it remains challenging to accurately pinpoint which amino acids will be in contact with the antigen (the paratope). In this work, we present a sequence-based probabilistic machine learning algorithm for paratope prediction, named Parapred. Parapred uses a deep-learning architecture to leverage features from both local residue neighbourhoods and across the entire sequence. The method outperforms the current state-of-the-art methodology, and only requires a stretch of amino acid sequence corresponding to a hypervariable region as an input, without any information about the antigen. We further show that our predictions can be used to improve both speed and accuracy of a rigid docking algorithm. The Parapred method is freely available at `https://github.com/eliberis/parapred` for download.**

I. INTRODUCTION

Antibodies are a special class of proteins produced by the immune system of vertebrates to neutralize pathogens, such as bacteria or viruses. They act by binding tightly to a unique molecule of the foreign agent, called the antigen. Antibody binding can mark it for future destruction by the immune system or, in some instances, neutralize it directly (*e.g.* by blocking a part of a virus essential for cell invasion). Typical antibodies are tetrameric—made of two immunoglobulin (Ig) heavy chains and two Ig light chains—and have a Y-shaped structure, where each of the two identical tips contains a binding site (paratope). The base of the Y mediates the ability of an antibody to communicate with other components of the immune system.

The paratope is typically contained within the hypervariable regions of the antibody which are also referred to as *complementarity determining regions (CDRs)*. In the structure of an antibody, CDRs are located within binding loops, three on each heavy chain (H1, H2, H3) and three on each light chain (L1, L2, L3). The variability of the CDR sequences allows antibodies to form complexes with virtually any antigen. This binding malleability of antibodies is increasingly harnessed by the biotechnological and biopharmaceutical industry; indeed, monoclonal antibodies are currently the fastest growing class of therapeutics on the market (Ecker *et al.* [1], Reichert [2]).

Novel antibodies that bind a target of interest can be obtained using well-established methods based on animal immunisation or on *in vitro* technologies for screening large laboratory-constructed libraries (Leavy [3]). However, for applications in research, diagnostics, and therapeutics, some degree of engineering is required to optimise certain properties, such as binding affinity, stability, solubility, or expression yield (Chiu *et al.* [4]). Rational engineering decisions become easier if detailed knowledge about an antibody under scrutiny is obtained (Chiu *et al.* [4], Sormanni *et al.* [5]). However, especially at the early stages of an antibody discovery campaign, only the sequence and an estimate of the binding affinity are usually available. Therefore, computational methods that can accurately predict molecular traits using just the amino acid sequence have a great potential for accelerating antibody discovery by assisting lead selection or facilitating property engineering.

For instance, hypervariable regions contain 40–50 amino acid residues, whereas typically less than 20 actually participate in binding (Esmaielbeiki *et al.* [6]), and some may even fall outside of the traditional definition of the CDRs (Kunik *et al.* [7]). The ability to accurately map the paratope would enable to pinpoint residues that are involved in binding, leaving others as candidate mutation sites that can be exploited to optimise other molecular traits, such as solubility or stability, without compromising the binding activity. In addition, as we show in this work, accurate paratope prediction can improve accuracy and speed of docking simulations, making structural models more reliable and easier to obtain.

In this work, we introduce the *Parapred* method for sequence-based prediction of paratope residues. Parapred improves on earlier methods for paratope prediction (Krawczyk *et al.* [9], Kunik *et al.* [7], Olimpieri *et al.* [8], Peng *et al.* [10], Tsuchiya & Mizuguchi [11]) by using deep learning methods and larger antibody datasets. Our method only requires the amino acid sequence of a CDR and four adjacent residues as its input, which, in contrast to structural data, can be readily obtained experimentally. For simplicity, we only consider antigens that are themselves proteins, which are the vast majority of known antibody targets.

"Deep learning" specifically refers to the process of building machine learning models consisting of multiple layers of non-linear operations, where each successive layer automatically learns more abstract representations (*features*) of the data using the features extracted by the previous layer (Goodfellow *et al.* [12, p. 1]). A key advantage of deep learning over traditional machine learning methods is that it can perform automated feature extraction directly from raw input data, thus eliminating the need for a domain expert to manually engineer features (Goodfellow *et al.* [12, p. 4]).

*Computer Laboratory, University of Cambridge, UK
†Department of Chemistry, University of Cambridge, UK
Contact e-mails: {el398, pv273, ps589, mv245, pl219}@cam.ac.uk

Automatically learned features are often found to be superior to manually engineered ones, contributing to the widespread success of deep learning in a range of fields. In particular, Parapred builds upon convolutional and recurrent neural networks, which achieved state-of-the-art results in object recognition (Krizhevsky *et al.* [13]) and machine translation (Wu *et al.* [14]) tasks, among others. Deep learning has already been successfully applied to address problems in protein science, including the prediction of structure (Li *et al.* [15]), function (Tavanaei *et al.* [16]) or binding sites (Alipanahi *et al.* [17]). To the best of our knowledge, this work is the first application of modern deep learning to antibody-antigen interactions.

II. METHOD

A. Data acquisition and preprocessing

To train and test our models, we used a subset of the Structural Antibody Database (SAbDab) (Dunbar *et al.* [18]), which contains antibody and antigen crystal structures. Entries in SAbDab were filtered (using the SAbDab web interface on 27 June 2017) to obtain a non-redundant set of antibody-antigen complexes with the following properties: (1) antibodies have both V_H and V_L domains present, (2) structure resolution is better than 3Å, (3) no two antibody sequences have $> 95\%$ sequence identity, (4) no two antigen sequences $> 90\%$ identical, and (5) each antibody has at least 5 residues in contact with the antigen. The final dataset contains 239 bound complexes (Supplementary Table C).

To construct the input, we identify the CDRs within the sequence of each antibody using the Chothia numbering scheme (Al-Lazikani *et al.* [19]). We augment the CDR sequences with two extra residues at both ends, as these residues are also known to sometimes engage in binding (Krawczyk *et al.* [9], Kunik *et al.* [7]). These extended CDR sequences are the input of the Parapred method and are processed individually.

Amino acid sequences have to be encoded as tensors prior to being processed by the model (Figure 1):

- Each amino acid sequence is encoded as a 'row' in a 3D matrix. As CDR sequences are usually of different length, they are padded with zero vectors to the length of the longest sequence. This is necessary for fast batch tensor processing provided by deep learning frameworks.
- Each element in a matrix encodes an amino acid residue and is itself a vector consisting of two concatenated parts:
  - *One-hot encoding* of the type of the residue (20 possible amino acid types + 1 extra, representing an unknown type). The type is encoded using a 21-dimensional vector, where all elements are set to 0 and one element, corresponding to the actual type of the amino acid, is set to 1.
  - Seven additional features, summarised by Meiler *et al.* [20], which represent physical, chemical and structural properties of each type of amino acid residue (Supplementary Table A).

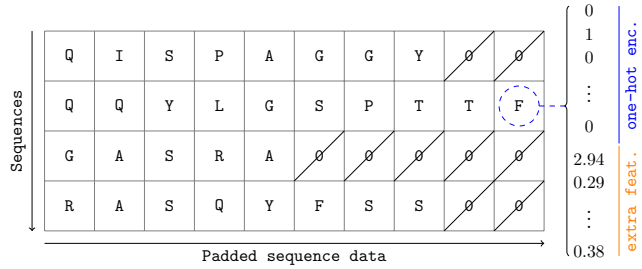

Fig. 1. An example of encoded amino acid sequences. An amino acid residue is represented by a feature vector which consists of one-hot encoding and some extra features. To cope with different sequence lengths, each sequence is padded to the length of the longest one.

The final dataset contains 1434 sequences for the algorithm to learn from (239 antibody/antigen complexes $\times$ 6 CDRs each).

B. Building a deep learning model

The paratope prediction problem can be formalised as a binary classification problem between two classes of residues: those that do not participate in binding (Class 0) and those that do (Class 1). Following previous conventions (Krawczyk *et al.* [9]), we define binding residues as those with at least one atom found within 4.5Å of any of the antigen atoms. Thus, for each residue in a sequence, the algorithm will output the probability p of it being in class 1 versus being in class 0 (the likelihood of binding).

Our model uses six prominent architectural developments in deep learning:

1) Multilayer perceptrons (MLP): Neural networks can be thought of as a set of interconnected units, called *neurons* or perceptrons, each of which performs a simple computation.

Neurons are typically arranged in *layers*, where each neuron in a layer is connected to the output of every neuron in the previous layer. A layer with this kind of connection is called *fully-connected*. The neural network itself is constructed as a series of such layers—the data is transformed in turn by every layer as it flows through the network. This architecture is known as a *deep feed-forward neural network* or a *multilayer perceptron* (MLP).

Neural network architectures are extensively used for machine learning tasks that can be reformulated as function approximation problems. We would like a network to learn to approximate some target function $f : X \to Y$ using a set of known input / output pairs for it (*supervised learning* setup). For example, for paratope prediction, $x \in X$ could be a vector encoding a residue and $y \in Y = \{0, 1\}$ could indicate whether the residue participates in binding.

The signals between neurons are real numbers and the neuron computes its output as follows:

- A neuron computes a weighted sum of its inputs ($\mathbf{x}$) and adds a constant term to it. The coefficients by which every input is scaled are called *weights* ($\mathbf{W}$) and the constant term is called the bias (b). The weights and bias constitute a set of adjustable *parameters* of a neuron.
- Some non-linear *activation function* σ is applied to the sum to produce the output. The activation function

introduces a non-linearity necessary to model complex functions.

We can compactly write the transformation performed by all neurons in a layer as a single weight matrix multiplication and bias vector addition:

$$\mathbf{y} = \sigma(\mathbf{W}^{\mathbf{T}}\mathbf{x} + \mathbf{b}) \tag{1}$$

2) Recurrent Neural Networks (RNN): We can design a neural network which processes every element in a sequence in turn. The key idea behind RNNs is to iteratively apply a simple processing block, called RNN *cell*, to obtain a summarised representation of a sequence up to any point. Figure 2 shows a computation graph of an RNN—the cell iteratively consumes inputs ($\mathbf{x}$) by computing a function of $\mathbf{x}$ and the previous state of the cell s.

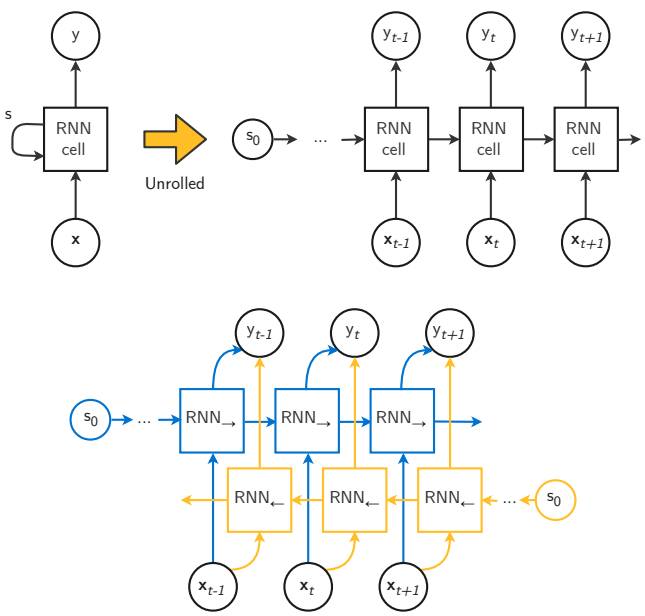

Fig. 2. The computation graph of RNNs used in this work. **Top:** the computation graph before (left) and after (right) unrolling. The same RNN cell is used to process every element of the input sequence. **Bottom:** unrolled graph of a bidirectional RNN. The inputs are passed through two different RNN cells (one for each direction) and the network's output at time t is an aggregation (here—concatenation) of the two cells' outputs.

We use the *Long Short-Term Memory (LSTM)* (Hochreiter & Schmidhuber [21]) cell which is able to learn long-range dependencies in sequences. The computation performed by an LSTM cell consists of the following steps:

- An LSTM cell holds the state s in two vectors: $\mathbf{C}$ ("memory") and $\mathbf{h}$ (previous output). Input $\mathbf{x}$ and state vector $\mathbf{h}$ are concatenated before being processed in four steps:

$$f_t = \sigma(\mathbf{W}_f^{\mathbf{T}}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{2}$$

$$C'_t = \tanh(\mathbf{W}_C^{\mathbf{T}}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \tag{3}$$

$$i_t = \sigma(\mathbf{W}_i^{\mathbf{T}}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{4}$$

$$o_t = \sigma(\mathbf{W}_o^{\mathbf{T}}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{5}$$

  where $\tanh$ is the element-wise hyperbolic tangent and σ is the logistic sigmoid function ($\sigma(x) = \frac{1}{1+\exp(-x)}$). Matrices $\mathbf{W}$ and vectors $\mathbf{b}$ are parameters learned by the network.
- The new cell state C_t and h_t, as well as the output y_t is given by:

$$C_t = C_{t-1} * f_t + C'_t * i_t \tag{6}$$

$$y_t = h_t = \tanh(C_t) * o_t \tag{7}$$

  where $*$ is the element-wise vector multiplication.

Capturing dependencies between an output and later inputs is necessary for amino acid sequences because they don't have a canonical direction (reading a sequence left to right is equivalent to reading it right to left). To achieve this, we use a *bidirectional RNN* (Schuster *et al.* [22]) which introduces a second pass going in the opposite direction (see Figure 2).

RNNs enable the model to capture features which span the entire input sequence.

3) Convolutional Neural Networks (CNN): Amino acid residues are known to interact with other residues and prefer some kinds of amino acids more than others as their neighbours (Xia *et al.* [23]). A paratope prediction model can exploit such preferences by processing every residue together with its neighbourhood to learn useful local patterns first and only then use an RNN to learn aggregate features of the entire sequence.

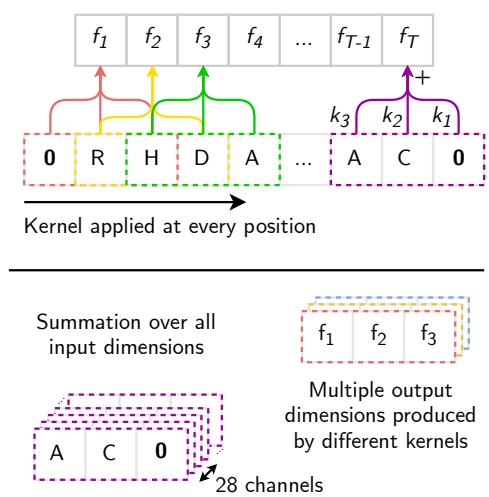

Fig. 3. An example of 1D convolution with kernel size 3. Outputs are computed by applying a kernel at each position in the input sequence.

Spatially local features can be extracted using *convolutional layers*, typically found in *convolutional neural networks (CNNs)*.

A convolutional layer is similar to a single-layer MLP discussed previously, only it uses a convolution operation instead of matrix multiplication. A convolution operation for sequences is defined as:

$$f_t = \sum_{i=-K'}^{K'} \mathbf{k}_{K'+1+i} \cdot \mathbf{i}_{t-i} \tag{8}$$

where $\mathbf{i}_t$ and f_t are elements of the input and output sequences at position t, respectively, and $\mathbf{k} \in \mathbb{R}^{K \times C}$ is a *kernel* of size $K = 2K' + 1$ (w.l.o.g assume that the kernel has an odd number of elements; C refers to the dimensionality of the input). This computation is visualised in Figure 3.

The kernel is applied this way at every position of the input sequence to produce the output sequence. For positions where kernel spans beyond the input sequence, we assume the input is padded with zero vectors: $\mathbf{i}_t = \mathbf{0}$ for $t \leq 0$ or $t > T$. The input and kernel elements themselves are vectors with multiple *channels*—name comes from an analogy with images: each pixel in an image has 3 dimensions: red, green and blue channels—*e.g.* an encoded residue would have 28 dimensions / channels (20 + 1 amino acid type one-hot encoding + extra 7 features, as described earlier).

Convolution performs a weighted summation over all dimensions of input elements to produce a single number (the sum of vector dot products). The fact that the same small kernel is applied to every position in the input sequence allows it to detect input patterns regardless of their position. Learnable parameters of a convolutional layer are its kernels; multiple output channels can be produced by using several different kernels (*filters*).

4) Residual Connections: Residual connections (He *et al.* [24]) act as a shortcut connection between inputs and outputs of some part of a network by adding inputs to outputs. Such shortcut can be added around the convolutional feature extractor—if the local feature extractor is supposed to learn some function $h(\mathbf{x})$, with the shortcut connection it only has to learn the residual $h(\mathbf{x}) - \mathbf{x}$ which is often easier to optimise for. The shortcut also enables the rest of the model to learn both from original inputs and extracted local features, and acts as a complexity controller by effectively allowing the network to adjust its depth.

5) Exponential linear units as activation functions: Activation functions introduce a non-linearity which is necessary to model complex functions. Experimenting with the activation function's behaviour can improve the training process. We use the *Exponential Linear Unit (ELU)* (Clevert *et al.* [25]) activation function which makes the network more robust to noise and faster to train. The function is given by:

$$ELU(x) = \begin{cases} x & \text{if } x \geq 0, \\ \alpha(e^x - 1) & \text{if } x < 0. \end{cases} \tag{9}$$

We use $\alpha = 1$.

6) Model regularisation: Deep learning models often have to be regularised to prevent overfitting—a phenomenon where a network memorises training examples (and noise) instead of modelling the underlying relationship. We use two regularisation methods:

- *Dropout* (Srivastava *et al.* [26]) is a computationally efficient regularisation method. The main idea of Dropout is to discard some intermediate results of the network at every training iteration with a certain probability p. This discourages the network from learning to rely on a particular subset of inputs.

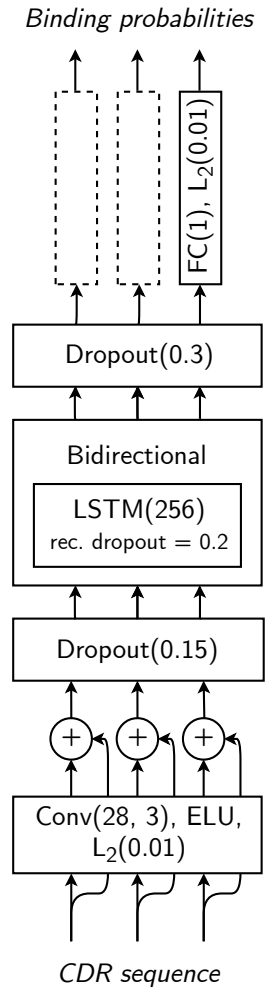

Fig. 4. The architecture of our paratope prediction model.

- L_2 regularisation (aka *weight decay*) adds an extra term—an L_2 norm of a layer's weights—to network's optimisation objective, which penalises weights if they grow too large during training.

C. Experimental setup

The software was developed in Python using TensorFlow deep learning framework (Abadi *et al.* [28]) and Keras API (Chollet *et al.* [27]). Overall, the network's computation consists of the following steps (Figure 4):

1) Encoded sequences (CDRs with 2 extra residues) are processed by a convolutional layer (regularised with an L_2 term scaled by 0.01) with 28 kernels, each spanning a neighbourhood of 3 residues. ELU activation is applied to the convolution results.
2) Residual connection is implemented by adding the original input sequences to the convolution output.
3) Resulting features are processed by a bidirectional LSTM with state size 256. The network applies Dropout with $p = 0.15$ to RNNs input and Dropout with $p = 0.2$ to RNNs recurrent connections.
4) Dropout with $p = 0.3$ is applied to the RNNs output and individual feature vectors are processed by

a single-output fully-connected network with logistic sigmoid activation function (to bring the output to the range of probabilities). Network's weights are regularised using an L_2 term scaled by 0.01.

The model's architecture could be easily augmented with layers that are able to process the 3D structure of an antibody in conjunction with its amino acid sequence. However, such sophisticated architectures would require a much larger training dataset (at least 10x more 3D structures) which is not available at this time. Training this kind of model would also require a way of efficiently exploiting cross-modality during feature extraction (Veličković *et al.* [34]).

Neural network training is a function optimisation problem, where we aim to find a local or global optimum of the optimisation target (aka *loss*) with respect to network's parameters. This should be a differentiable measure of how well the neural network approximates the target function. We use the *binary cross-entropy* loss, a popular choice for binary classification problems:

$$\mathcal{L}(\Theta) = \frac{1}{m} \sum_{\mathbf{x}_i, y_i \in \mathtt{TrS}} w_i^s (-y_i \log(\hat{f}_\Theta(\mathbf{x}_i)) - (1 - y_i) \log(1 - \hat{f}_\Theta(\mathbf{x}_i))) \quad (10)$$

where $\mathtt{TrS}$ is the training set of size m, $\hat{f}_\Theta$ is the function computed by the network with parameters Θ and w_i^s is the sample weight (described later).

To find a loss minima, we use the *Adam* (Kingma & Ba [29]) optimiser with base learning rate setting of 0.01 for the first 10 epochs and 0.001 otherwise. The network is trained with 32 samples at once (aka *batch size*) for 16 epochs (iterations over the entire training set).

The dataset has an uneven number of binding (positive) and non-binding (negative) residues—3.4x more negative samples. The cross-entropy loss function (Equation 10) equally penalises misclassified positive and negative samples, which allows the model to keep the overall loss low by preferring to predict that residues will not bind. This achieves good classification accuracy but hinders the model's ability to learn to identify positive samples. This can be improved by penalising misclassified positive samples more—the per-sample loss is scaled by the *sample weight* w_i^s which we set to a 2.5x higher value for positive samples.

III. RESULTS

A. Model results

We used the *10-fold cross-validation* technique to assess the model performance on multiple dataset splits. This technique randomly partitions the data into ten subsets and trains the model ten times. Each time, a different subset is selected for testing, and the method is trained on the sequences belonging to the other nine. Because the results may vary between cross-validation runs due to the initial random partition of the data, the random initialisation of the network parameters, and Dropout, the 10-fold cross-validation is repeated ten times, which also enable to calculate confidence interval of the mean values of each performance indicator.

To measure the performance of the binary classifier we use a number of standard metrics:

- **Recall:** (Also known as *sensitivity* or *true positive rate*.) The proportion of positive samples classified correctly (= $\frac{TP}{TP+FN}$).
- **Precision:** (Also known as *positive predictive value*.) The proportion of actual positive samples among all samples predicted to be positive (= $\frac{TP}{TP+FP}$).
- **F-score:** Precision and recall measure two independent properties of a classifier—F-score combines both of them by measuring their harmonic mean.
- **Matthews correlation coefficient (MCC):** MCC is a popular measure of binary classification quality for unbalanced datasets.
- **Precision-recall (PR) curve:** Aforementioned metrics require a particular classification threshold. However, the user can vary the threshold to adjust the confidence of results they get. For example, setting the threshold to a high value would label only very few residues as positive (those that the model is very certain about), so the classification would have low recall but high precision. Conversely, setting the threshold to a low value would yield high recall but low precision results. To visualise this relationship, we plot precision at different levels of recall obtained by varying the threshold.
- **Receiver operating characteristic (ROC) curve:** ROC curves show the relationship between the true positive rate (recall) and the false positive rate (the proportion of false positives among all negative samples, $\frac{FP}{FP+TN}$), similarly obtained by varying the threshold. Area under the ROC curve (**ROC AUC**) is often used to compare classifiers.

Table 1 shows the F-score, MCC and ROC AUC performance metrics of Parapred. Narrow confidence intervals indicate consistent performance across cross-validation rounds. Furthermore, the results show that our model outperforms the current state of the art predictor, *proABC* (Olimpieri *et al.* [8]), without needing the entire antibody sequence or extra features such as the germline family or antigen volume.

Model	F-score	MCC	ROC AUC
Parapred	0.705 ± 0.003	0.572 ± 0.005	0.883 ± 0.002
proABC	—	0.522	0.851

Table 1. Performance indicators of the Parapred method with 95% confidence intervals (top row) and of the proABC method (bottom row). The F-score and MCC metrics use a classification threshold of 0.4913739, obtained by maximising Youden's index (Youden [30]).

Figure 5 shows the precision/recall curve of the Parapred method (in blue) which indicates a substantial improvement over Antibody i-Patch (orange points). This improvement is particularly relevant given that, in contrast to Parapred, Antibody i-Patch requires a structure or a homology model of the antibody and the antigen it binds to.

We investigated to what extent the performance improvement originates from using a larger dataset (239 complexes vs. 148 of Antibody i-Patch) and to what from the deep-

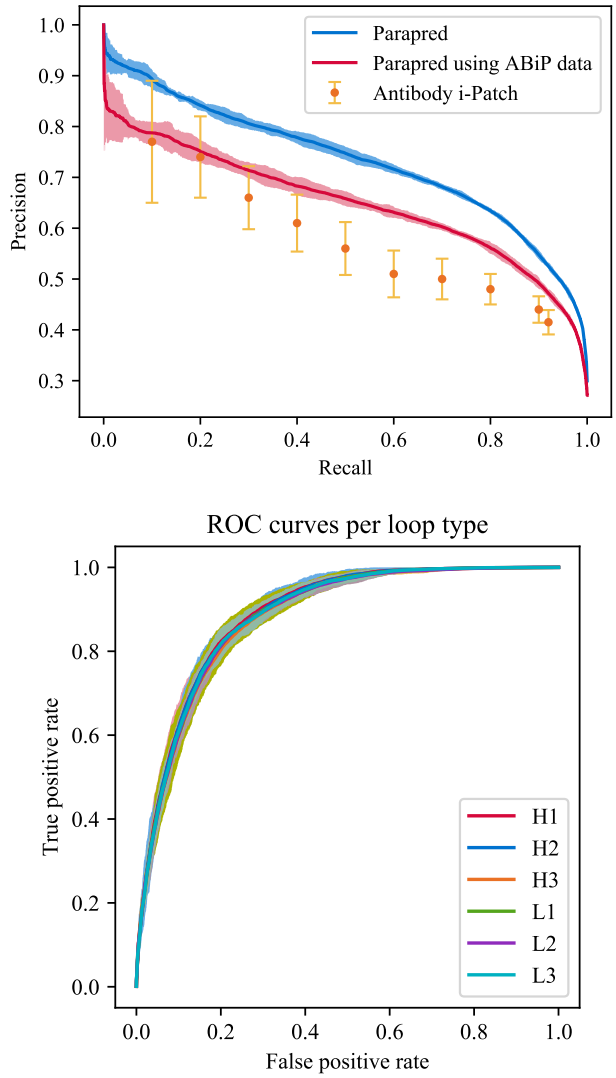

Fig. 5. Parapred performance curves. **Top:** precision-recall curves of Parapred, obtained from 10-fold cross-validation runs, when trained on our dataset (blue) and Antibody i-Patch's dataset (red), together with the PR values reported by the authors of Antibody i-Patch (orange). **Bottom:** ROC curves of Parapred, separated by loop type. Shaded areas / bars show 95% confidence bounds (2 standard errors).

learning-based architecture of Parapred. To assess this, we measured Parapred's performance when trained on the Antibody i-Patch's dataset (red curve in Figure 5). We find that our method achieves significant precision improvements for recall values > 0.5, which is typically the most useful range. We conclude that the deep-learning-based architecture of Parapred is able to capture a richer set of features leading to better classification, even though it uses less explicit information about the antibody (Parapred does not require structural data or any information about the antigen it binds to). The leap in performance from the red to the blue curve is in agreement with the observation that deep models thrive in environments with a larger number of data points to learn from (Goodfellow *et al.* [12, p. 430]).

Our encoding of an amino acid sequence does not include information about the CDR loop type it originated from, so the model would not be able to capture loop type-specific features. Figure 5 shows the ROC curves of our model's predictions, separated by CDR types. The graph shows that the model is able to predict binding residues equally well for all six CDR types despite lacking loop type information. We found that including the loop type made no appreciable difference to the performance.

B. Docking improvements

We show the usefulness of Parapred by integrating its predictions with the PatchDock rigid protein docking algorithm (Duhovny *et al.* [31]).

PatchDock works with 2 protein molecules in the PDB format and searches for suitable orientations for one of the molecules—conventionally, the antigen—"onto" the other, such that the two form an antibody-antigen complex. The algorithm produces several hundred candidate orientations of the antigen, called *decoys*, which are ranked in the output by an internal scoring function. The algorithm also provides facility to guide the search process by pre-specifying potential binding site residues.

Decoys can be stratified into 4 quality classes—high (***), medium (**), low (*) or unclassed—based on how close the computed orientation of the antigen is to the true (*native*) orientation recorded in the dataset. The classification uses the CAPRI criteria (see Appendix B).

The usefulness of Parapred was measured by running PatchDock with three potential binding sites of the antibody molecule: (1) the CDRs, (2) the actual paratope and (3) binding residues predicted by Parapred (residues with > 0.67 binding probability to match the number of residues in the actual paratope). We picked 30 antibody-antigen complexes at random (highlighted in Appendix C) to be run through the docking algorithm and used the remaining 209 to train the model.

We recorded the best class decoy in the top 10 and top 200 decoys, as ranked by PatchDock, for each of the 30 structures. As a hint, we also supplied the antigen's binding site as residues within 5Å of the real epitope. PatchDock was run with default parameters.

Binding site	**Top 10**			**Top 200**		
	***	**	*	***	**	*
CDRs	0	2	0	1	14	0
Paratope	0	7	1	1	21	3
Parapred	0	8	0	1	19	2

Table 2. The number of high, medium and low quality decoys obtained by running PatchDock with different constraints on a test set of 30 structures.

Docking results are shown in Table 2—supplying just the CDR gives the worst performance, however, supplying our model's predictions achieves performance on par with supplying the actual paratope. We conclude that for docking simulations Parapred's predictions are almost as informative as the actual paratope.

We also measured the time taken by PatchDock to produce decoys on a machine with an "Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz" processor (Table 3). We found

Binding site	Running time
CDRs	3h 50min 19.72s
Paratope	2h 02min 22.50s
Parapred	2h 11min 15.40s
(1) vs (3) speedup:	1.75x

Table 3. PatchDock running time using different binding site constraints.

that specifying Parapred's predictions as a potential binding site produces a 1.7x speedup in PatchDock's computations compared to specifying just the CDRs.

C. Interpreting local neighbourhood features

The use of RNN allows our model to learn complex long-range dependencies between residues. This expressiveness also comes at a cost of interpretability—it is difficult to identify precisely which factors the model considers to be the most important. Nevertheless, we can inspect the model's output and attempt to relate it to known findings to further confirm them or identify the model's biases.

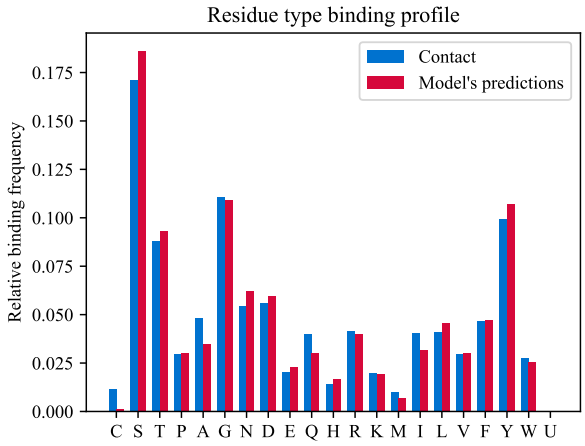

Fig. 6. Frequency of residues of a particular type participating in binding, as computed from the dataset (blue) and as predicted by our model (red)

Figure 6 shows how frequently each residue type participates in binding, based on contact information from the dataset and our model's predictions. Residue was considered to be binding if it had > 0.5 mean binding probability according to crossvalidation results. Overall, the model approximates the true binding profile well, showing particular preference for serine (S), tyrosine (Y), glycine (G) and threonine (T). Interestingly, the model largely ignores cysteine (C), most likely due to its tendency to form disulphide bonds and not therefore participate in binding.

As discussed in the previous section, the first layer of our network consists 28 convolutional filters with kernel width 3, acting as local feature extractors. We can inspect what these filters learned by investigating which sequences of length 3 result in their highest response (*activation*). We measure activations after the the residual connection adds one-hot encoded residue type to the convolutional layer's output; this allows to include the activation boost of each residue type for the first 20 filters due to one-hot encoding.

Table 4 shows neighbourhoods with highest activations for the first 20 filters (corresponding to each residue type). Findings include the following interesting points:

- Tryptophan (W) is present in many residue neighbourhoods, even though it does not often participate in binding itself.
- Kernel for lysine (K) does not actually consider lysine to be the most informative residue. It learns arginine's (R) neighbourhoods.

This investigation into model interpretability could be further extended by adding attention layers to the model (Bahdanau *et al.* [35]), which would allow inspecting positions in the input sequence that model learned to relate the most.

IV. CONCLUSIONS

A. Summary

To our best knowledge, this work is the first application of modern deep learning (CNN- and RNN-based neural networks) to the paratope prediction problem. Our model is able to generalise using only antibody sequence stretches corresponding to the CDRs (with 2 extra residues on the either side) and outperforms the current state-of-the-art by a statistically significant margin. We also showed that the model's predictions provide speed and quality gains for the PatchDock rigid docking algorithm—decoy quality and time-to-dock were comparable to ones obtained when the docking algorithm has knowledge of the CDRs.

One of the main benefits of Parapred is that it does not rely on any higher-level antibody features: no full sequence, homology model, crystal structure or antigen information is required. However, if the user has such data already available, they might be able to usefully integrate it with the model, for example, by adding extra per-residue features.

REFERENCES

[1] Ecker, D. M., Jones, S. D., & Levine, H. L. (2015). The therapeutic monoclonal antibody market. *MAbs*, 7(1), pp. 9–14. Taylor & Francis.

[2] Reichert, J. M. (2017). Antibodies to watch in 2017. *MAbs* 9(2), pp. 167–181. Taylor & Francis.

[3] Leavy, O. (2010). Therapeutic antibodies: past, present and future. *Nature Reviews Immunology*, 10(5), p. 297.

[4] Chiu, M. L., & Gilliland, G. L. (2016). Engineering antibody therapeutics. *Current opinion in structural biology*, 38, pp. 163–173.

[5] Sormanni, P., Amery, L., Ekizoglou, S., Vendruscolo, M., Popovic, B. (2017). Rapid and accurate in silico solubility screening of a monoclonal antibody library. *Scientific Reports*, 7, a.n. 8200.

[6] Esmaielbeiki, R., Krawczyk, K., Knapp, B., Nebel, J. C., & Deane, C. M. (2016). Progress and challenges in predicting protein interfaces. *Briefings in bioinformatics*, 17(1), pp. 117–131.

[7] Kunik, V., Peters, B., & Ofran, Y. (2012). Structural consensus among antibodies defines the antigen binding site. *PLoS Comput. Biol.*, 8(2), e1002388.

[8] Olimpieri, P. P., Chailyan, A., Tramontano, A., & Marcatili, P. (2013). Prediction of site-specific interactions in antibody-antigen complexes: the proABC method and server. *Bioinformatics*, btt369.

[9] Krawczyk, K., Baker, T., Shi, J., & Deane, C. M. (2013). Antibody i-Patch prediction of the antibody binding site improves rigid local antibody–antigen docking. *Protein Engineering Design and Selection*, 26(10), pp. 621–629.

0/C	1/S	2/T	3/P	4/A	5/G	6/N	7/D	8/E	9/Q	10/H	11/R	12/K	13/M	14/I	15/L	16/V	17/F	18/Y	19/W
UCR	TSI	WTW	RPI	UAG	WGI	INU	UDW	UER	UQR	RHI	WRU	WRR	UMR	VIW	ULR	WVU	GFG	UYW	WWE
UCK	TSV	WTE	RPV	UAK	WGU	ING	VDW	UEK	GQR	KHI	YRU	WRK	UMK	TIW	ULW	WVG	GFA	UYC	WWU
UCC	ASI	WTY	KPI	UAR	WGV	IND	IDW	UEW	UQK	RHV	FRU	YRR	GMR	KIW	ULY	YVU	KFG	UYF	WWD
UCG	TSC	WTR	KPV	UAA	RGI	VNU	DDW	UWR	DQR	KHV	WRG	FRR	GMK	PIW	ULK	RVU	AFG	UYU	WWW
UCA	PSI	WTF	RPT	UAS	RGU	INA	LDW	UEY	EQR	RHT	WRD	YRK	UMH	RIW	ULF	WVA	GFU	UYG	WWY
UCP	ASV	WTQ	KPT	UAC	WGT	INE	MDW	DER	AQR	RHP	RRU	FRK	UMP	AIW	ULQ	WVS	GFC	DYW	WWF
UCH	TSP	WTD	RPP	UAN	RGV	INN	CDW	UEH	SQR	RHL	WRE	WRH	UMV	IIW	ULE	FVU	GFS	UYL	WWQ
UCS	TSA	WTK	KPP	UAQ	KGI	INS	PDW	EER	UQW	KHT	WRS	WRW	AMR	HIW	ULM	WVD	KFA	DYC	WWM
UCN	VSI	WTM	RPU	UAU	WGP	INM	TDW	UEF	NQR	RHW	WWU	ERR	DMR	GIW	ULH	YVG	RFG	UYM	WWN
UCL	ISI	WTH	KPU	DAG	WGD	INC	FDW	GER	GQK	KHP	WRA	MRR	UMN	SIW	ULL	RVG	GFP	EYW	WWL

Table 4. Top 10 sequences that activate the first 20 convolutional filters the most, sorted in descending order of the absolute value of activation. The presence of `U` indicates that the filter does not have a particular neighbour preference.

[10] Peng, H. P., Lee, K. H., Jian, J. W., & Yang, A. S. (2014). Origins of specificity and affinity in antibody–protein interactions. *Proceedings of the National Academy of Sciences*, 111(26)E, pp. 2656–2665.

[11] Tsuchiya, Y., & Mizuguchi, K. (2016). The diversity of H3 loops determines the antigen-binding tendencies of antibody CDR loops. *Protein Science*, 25(4), pp. 815–825.

[12] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. *MIT Press*.

[13] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pp. 1097–1105.

[14] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., …& Klingner, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint*, arXiv:1609.08144.

[15] Li, R., Si, D., Zeng, T., Ji, S., & He, J. (2016). Deep convolutional neural networks for detecting secondary structures in protein density maps from cryo-electron microscopy. *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 41–46.

[16] Tavanaei, A., Maida, A. S., Kaniymattam, A., & Logananthara j, R. (2016). Towards recognition of protein function based on its structure using deep convolutional networks. *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 145–149.

[17] Alipanahi, B., Delong, A., Weirauch, M. T., & Frey, B. J. (2015). Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning (DeepBind). *Nature biotechnology*, 33(8), pp. 831–838.

[18] Dunbar, J., Krawczyk, K., Leem, J., Baker, T., Fuchs, A., Georges, G., …& Deane, C. M. (2014). SAbDab: the structural antibody database. *Nucleic acids research*, 42(D1), D1140-D1146.

[19] Al-Lazikani, B., Lesk, A. M., & Chothia, C. (1997). Standard conformations for the canonical structures of immunoglobulins. *Journal of molecular biology*, 273(4), pp. 927–948.

[20] Meiler, J., Müller, M., Zeidler, A., & Schmäschke, F. (2001). Generation and evaluation of dimension-reduced amino acid parameter representations by artificial neural networks. *Molecular modeling annual*, 7(9), pp. 360–369.

[21] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), pp. 1735–1780.

[22] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), pp. 2673–2681.

[23] Xia, X., & Xie, Z. (2002). Protein structure, neighbor effect, and a new index of amino acid dissimilarities. *Molecular biology and evolution*, 19(1), pp. 58–67.

[24] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5, pp. 6–14.

[25] Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint*, arXiv:1511.07289.

[26] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), pp. 1929–1958.

[27] Chollet, F. and others. (2015). Keras. *GitHub*. `https://github.com/fchollet/keras`

[28] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint*, arXiv:1603.04467.

[29] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint*, arXiv:1412.6980.

[30] Youden, W. J. (1950). Index for rating diagnostic tests. *Cancer*, 3(1), pp. 32–35.

[31] Duhovny, D., Nussinov, R., & Wolfson, H. J. (2002). Efficient unbound docking of rigid molecules. *International Workshop on Algorithms in Bioinformatics*, pp. 185–200.

[32] Janin, J., Henrick, K., Moult, J., Eyck, L. T., Sternberg, M. J., Vajda, S., ... & Wodak, S. J. (2003). CAPRI: a critical assessment of predicted interactions. *Proteins: Structure, Function, and Bioinformatics*, 52(1), pp. 2–9.

[33] Méndez, R., Leplae, R., De Maria, L., & Wodak, S. J. (2003). Assessment of blind predictions of protein–protein interactions: current status of docking methods. *Proteins: Structure, Function, and Bioinformatics*, 52(1), pp. 51–67.

[34] Veličković, P., Wang, D., Lane, N. D., & Liò, P. (2016). X-CNN: Cross-modal convolutional neural networks for sparse datasets. *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8.

[35] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint*, arXiv:1409.0473.

APPENDIX

A. Extra features in the antibody residue encoding

See Table 5.

B. Decoy classification

Decoys are classified using the CAPRI criteria (Janin *et al.* [32], Méndez *et al.* [33]):

f_{NAT}: Two residues, one from the antibody and one from the antigen, are said to be an *interface pair* if they are sufficiently close to interact (here residues are assumed to be interacting if they are less than 5Å apart) . f_{NAT} is a proportion of interface pairs in the native complex that were reproduced in the generated one.

L_{RMS}: The root mean square deviation (RMSD) of atoms' coordinates in the generated orientation of the antigen compared to the true one (the lower the better). For two sets of coordinates $\mathbf{u}$ and $\mathbf{v}$ of size n, the RMSD is given by:

$$\text{RMSD}(\mathbf{u}, \mathbf{v}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \|u_i - v_i\|^2} \qquad (11)$$

The RMSD is computed only for *backbone heavy atoms* (N, C, C_α, O).

I_{RMS}: The L_{RMS} measure unnecessarily penalises large antigen chains when the epitope has a roughly

AA	Ξ	α_p	ν_v	π	I	α	β
C	1.77	0.13	2.43	1.54	6.35	0.17	0.41
S	1.31	0.06	1.60	-0.04	5.70	0.20	0.28
T	3.03	0.11	2.60	0.26	5.60	0.21	0.36
P	2.67	0.00	2.72	0.72	6.80	0.13	0.34
A	1.28	0.05	1.00	0.31	6.11	0.42	0.23
G	0.00	0.00	0.00	0.00	6.07	0.13	0.15
N	1.60	0.13	2.95	-0.60	6.52	0.21	0.22
D	1.60	0.11	2.78	-0.77	2.95	0.25	0.20
E	1.56	0.15	3.78	-0.64	3.09	0.42	0.21
Q	1.56	0.18	3.95	-0.22	5.65	0.36	0.25
H	2.99	0.23	4.66	0.13	7.69	0.27	0.30
R	2.34	0.29	6.13	-1.01	10.74	0.36	0.25
K	1.89	0.22	4.77	-0.99	9.99	0.32	0.27
M	2.35	0.22	4.43	1.23	5.71	0.38	0.32
I	4.19	0.19	4.00	1.80	6.04	0.30	0.45
L	2.59	0.19	4.00	1.70	6.04	0.39	0.31
V	3.67	0.14	3.00	1.22	6.02	0.27	0.49
F	2.94	0.29	5.89	1.79	5.67	0.30	0.38
Y	2.94	0.30	6.47	0.96	5.66	0.25	0.41
W	3.21	0.41	8.08	2.25	5.94	0.32	0.42
U	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 5. Extra features used in the amino acid (AA) residue encoding (U stands for the extra 'unknown' type): Ξ—steric parameter, α_p—polarisability, ν_v—volume, π—hydrophobicity, I—isoelectric point, α—helix probability, β—sheet probability.

correct orientation but the rest of the chain does not. To better handle such cases, the I_{RMS} measure was introduced—the RMSD only of residues within 10Å of the epitope.

Decoys are assigned a class according to criteria in Table 6.

Class	f_{NAT}	L_{RMS}	I_{RMS}
High	≥ 0.5	≤ 1.0	or ≤ 1.0
Medium	≥ 0.3	$1.0 < \cdot \leq 5.0$	or $1.0 < \cdot \leq 2.0$
Acceptable	≥ 0.1	$5.0 < \cdot \leq 10.0$	or $2.0 < \cdot \leq 4.0$
Incorrect	< 0.1	—	—

Table 6. Decoy classification using f_{NAT}, L_{RMS} and I_{RMS} measures. The decoy has to match the f_{NAT} criteria and either the L_{RMS} or I_{RMS} criteria.

C. Dataset

See Table 7.

PDB	V_H	V_L	Ag	PDB	V_H	V_L	Ag	PDB	V_H	V_L	Ag	PDB	V_H	V_L	Ag
1bgx	H	L	T	3grw	H	L	A	4dkf	H	L	A	4wfe	G	F	B
1egj	H	L	A	3h42	H	L	BA	4dtg	H	L	K	4wv1	E	D	F
1eo8	H	L	A	3hb3	C	D	B	4dvr	H	L	G	4xmp	H	L	G
1fns	H	L	A	3hi6	X	Y	B	4dw2	H	L	U	4xp4	H	L	A
1fsk	C	B	A	3idx	H	L	G	4edw	H	L	V	4xtr	C	D	B
1h0d	B	A	C	3iu3	H	L	I	4etq	H	L	C	4xvs	H	L	G
1jrh	H	L	I	3k2u	H	L	A	4f37	H	L	A	4y5y	A	B	C
1kb5	H	L	AB	3kr3	H	L	D	4f3f	B	A	C	4ye4	H	L	G
1lk3	H	L	A	3ks0	H	L	B	4ffv	D	C	B	4ywg	H	L	G
1ncb	H	L	N	3l5x	H	L	A	4ffy	H	L	A	4zfg	H	L	A
1nsn	H	L	S	3l95	H	L	Y	4fqj	H	L	A	4zso	D	C	F
1oaz	J	N	B	3ldb	C	B	A	4g7v	H	L	S	5anm	B	A	G
1ors	B	A	C	3lev	H	L	A	4h88	H	L	A	5b3j	E	F	C
1r0a	H	L	B	3lh2	I	M	T	4hcr	M	N	B	5b71	B	A	E
1rjl	B	A	C	3liz	H	L	A	4hkx	A	B	E	5bo1	I	M	A
1sy6	H	L	A	3lzf	H	L	A	4hlz	G	H	BA	5bv7	H	L	A
1tpx	B	C	A	3ma9	H	L	A	4ht1	H	L	T	5czx	H	L	A
1v7m	H	L	V	3mj9	H	L	A	4hwb	H	L	A	5d96	J	I	D
1w72	H	L	AC	3mxw	H	L	A	4i2x	B	A	E	5dfv	C	D	A
1wej	H	L	F	3nh7	J	N	C	4i3s	H	L	G	5dhv	H	L	M
1xiw	D	C	AF	3nps	B	C	A	4ij3	C	B	A	5do2	C	D	B
1yjd	H	L	C	3o2d	H	L	A	4j4p	C	D	B	5e8e	B	A	H
1ztx	H	L	E	3pgf	H	L	A	4j6r	H	L	G	5e94	B	A	G
2adf	H	L	A	3pnw	H	G	I	4jlr	H	L	S	5en2	A	B	C
2arj	H	L	Q	3q1s	H	L	I	4jpk	H	L	A	5eu7	E	C	A
2b2x	I	M	B	3q3g	H	F	I	4jqi	H	L	AV	5f3b	E	F	D
2bdn	H	L	A	3r1g	H	L	B	4jr9	H	L	A	5fb8	B	A	C
2dd8	H	L	S	3ru8	H	L	X	4k2u	I	M	B	5fcu	H	L	G
2fd6	H	L	U	3s35	H	L	X	4k3j	H	L	B	5ggs	A	B	Z
2h9g	B	A	R	3sdy	H	L	BA	4k94	H	L	CC	5ggt	H	L	A
2ih3	A	B	C	3sob	H	L	B	4ki5	E	F	M	5ggv	H	L	Y
2j88	H	L	A	3t2n	I	M	B	4kuc	F	E	I	5gjs	H	L	BA
2jel	H	L	P	3u30	C	B	A	4kxz	J	I	B	5h35	H	I	D
2nyy	D	C	A	3u4e	A	B	J	4liq	H	L	E	5hbt	D	C	B
2oz4	H	L	A	3uc0	I	M	B	4lmq	H	L	D	5hi4	C	D	B
2q8b	H	L	A	3v6o	C	E	A	4lqf	H	L	A	5ikc	H	L	N
2qqk	H	L	A	3vg9	C	B	A	4lsp	H	L	G	5j13	C	B	A
2qqn	H	L	A	3vi4	F	E	B	4lst	H	L	G	5jq6	H	L	A
2r56	I	M	B	3w9e	A	B	C	4lvh	B	C	A	5kw9	H	L	A
2vxt	H	L	I	3wfd	H	L	BC	4lvn	C	B	A	5l0q	F	E	D
2xqb	H	L	A	3wih	I	M	B	4m62	I	M	T	5lqb	H	L	A
2xqy	G	L	A	3wkm	I	M	B	4np4	H	L	A	5lxg	H	L	A
2yc1	D	E	F	3zkm	C	D	B	4nzt	H	L	M	5mvz	H	L	U
2zch	H	L	P	4aei	I	M	B	4ogy	M	N	B	5te7	H	L	G
3b2u	T	U	S	4ag4	H	L	A	4okv	C	D	E	5th9	H	L	A
3b9k	D	C	F	4ala	H	L	C	4ot1	H	L	A	5tl5	H	L	A
3cvh	H	L	AC	4bz1	H	L	A	4plj	D	C	B	5tq0	H	L	AB
3cx5	J	K	E	4cad	K	J	L	4r8w	H	L	BA	5tz2	H	L	C
3d85	B	A	C	4cmh	B	C	A	4rdq	I	H	A	5v7j	H	L	G
3gbm	I	M	DC	4cni	H	L	C	4rgn	F	G	S	5v7j	D	E	B
3gi9	H	L	C	4d3c	H	L	A	4u6v	H	L	A	5veb	A	B	X
3gjf	H	L	AC	4d9q	E	D	B	4uu9	A	B	C	5vpg	D	C	A
												5wux	A	B	F
2uzi	H	L	R	**5gzo**	C	D	B	**3skj**	H	L	E	**4rrp**	K	E	Q
2ypv	H	L	A	**4leo**	A	B	C	**3rkd**	H	L	A	**3u9p**	K	M	C
5k59	H	L	A	**4nx3**	B	A	D	**4qww**	D	C	B	**1osp**	H	L	O
1jps	H	L	T	**5tih**	H	L	A	**4qci**	H	L	C	**1ob1**	B	A	C
2xwt	A	B	C	**2vxq**	H	L	A	**4irz**	H	L	A	**4ene**	E	F	B
3d9a	H	L	C	**1nfd**	F	E	B	**3t3p**	E	F	C	**5d70**	H	L	A
2aep	H	L	A	**4mwf**	H	L	C	**4jzj**	A	B	D	**5mes**	H	L	A
4nnp	H	L	A	**5mi0**	B	C	A								

Table 7. 239 bound antibody-antigen complex dataset used to train our model. Each entry shows the PDB code, as well as antibody heavy, antibody light and antigen chain IDs. Structures in bold were used as a test set for the docking experiment.