Chapter 1

A Recurrent Neural Network to Predict G Quadruplex Structure

1.1 Introduction

Because of the dependence of G4 structure largely on sequence information, it is possible to make predictions about the propensity of specific sequences to form G4s using pattern matching analyses. The initial rule which was employed for putative G4 (PG4) detection in the human genome was the Quadparser method (Huppert and Balasubramanian, 2005). This is a simple regular expression following the pattern $G_X N_{1-7} G_X N_{1-7} G_X N_{1-7} G_X$ where $X \geqslant 3$. The Quadparser method was chosen based upon early Circular Dichroism and UV melting data, which suggested that G4s tended to be more stable with three or more tetrads, relatively short loop lengths, and no bulges in tetrads. These fairly stringent rules for formation mean that in general, the Quadparser method is considered to be fairly conservative, and misses a lot of sequences with high G4 forming potential.

More recently there has been a large increase in the number of available methods for predicting G4s (Bedrat et al., 2016; Garant et al., 2017; Hon et

al., 2017; Sahakyan et al., 2017). The contribution from Bedrat et al., named G4Hunter, is a scoring method based on a run length encoding the input sequence. Runs of Gs score positively whilst runs of Cs score negatively. It can be used with a sliding window approach to score an entire genome, with thresholding to identify high scoring PG4s. Whilst this approach is much more tolerant of imperfections which violate the Quadparser method, it is arguably too tolerant, producing many false positives. Furthermore, it does not take into account flanking A and T sequences which may contribute to the stability of the G4, e.g. through reducing the favourability of double stranded DNA.

Some middle ground is required to improve existing methods. The current global interest in machine learning for solving biological problems has recently been brought to the field of G4 research, to attempt to solve this issue. One G4 prediction tool, G4RNA screener, is a densely connected neural network which is trained on the trinucleotide contents of input sequences (Garant et al., 2017). This model was trained by Garant et al. using a set of melting temperatures of RNA G4s obtained from a literature search. Currently this database contains only 368 sequences, however. Given the almost inconceivable number of potential G4 forming sequences (there are more than 10^{12} sequences which could match the original Quadparser pattern, not including flanking sequences), it is probable that this dataset does not capture all of the variety of possible RNA G4 forming sequences. To do this, a more high throughput method for measuring G4 forming propensity is required.

A new method for sequencing of genomic G4 structures, termed G4Seq, may provide this level of throughput (Chambers et al., 2015). To create this dataset, Chamber et al. sequenced human genomic DNA in an Illumina sequencing-by-synthesis machine, in the presence or absence of G4 stabilising potassium cations, or G4 binding ligand Pyridostatin. The G4 structures caused stalling of DNA polymerase, resulting in a large number of errors in the resulting read. When the authors then mapped the resultant reads, the mismatch rate which

occurred at each position in the genome could be counted to create a map of G4-forming loci. Another high throughput method for sequencing G4s, this time in human mRNAs, was developed by Kwok et al. This protocol utilises the ability of potassium or pyridostatin stabilised RNA G4s to stall reverse transcriptase, resulting in a drop off of reads in the 3'->5' direction in RNAseq data (Kwok et al., 2016).

Data from the G4Seq experiment was leveraged by Sahakyan et al. to build an extreme gradient boosted machine model, named Quadron, to predict G4 formation (Sahakyan et al., 2017). Quadron initially predicts PG4s using the Quadparser method, but with extended loop lengths of 1-12. Various derived features, such as tetrad number, loop length and mono-, di- and trinucleotide content, are then extracted from these patterns and used to train a model predicting G4Seq mismatch rate. The primary drawback of this method is that the Quadparser method is still used to make initial predictions. This means that Quadron is only able to improve the precision of the existing Quadparser method by rejecting false positives that do not form G4s. Since the Quadparser method is already quite conservative, a lot of potential G4 forming sequences are still missed.

Here we present a new method for G4 prediction, which builds on the work of Bedrat et al. and Sahakyan et al. We use the G4Seq dataset as training data, however a convolutional and recurrent neural network is used to process input sequences directly, meaning fewer prior assumptions are required about what constitutes a G4 forming sequence. Our new method, which we name G4Seeqer, performs better on both the G4Seq datasets and other datasets, including G4s immunoprecipitated from chromatin (Hänsel-Hertsch et al., 2016). We also use transfer learning to apply the model learned from G4Seq data to RNA G4 prediction. Finally, we use G4Seeqer to characterise unnoticed features of the G4Seq dataset, and compare our scores to data from UV melting experiments.

1.2 Materials and Methods

1.2.1 G4Hunter Algorithm

The G4hunter algorithm from Bedrat et al. was reimplemented in Cython (Python superset which can be compiled to C) with some alterations (Bedrat et al., 2016). Input sequences were run length encoded, and each run of Gs was scored as the square of length of the run, with a maximum score per run of 16. These scores were summed to give a positive strand total score. Runs of Cs were scored equivalently but in a separate negative strand score. Scores were divided by the length of the input sequence to get a normalised score.

1.2.2 Training Data Preprocessing

The modified G4Hunter method was run on the hg19 genome using a window size of 50bp, a step size of 5 and a threshold of 0.75 to generate a total of 7484506 candidate G4 intervals, which were output in bed format. Intervals were increased in size by 39bp in each direction using `bedtools slop` to introduce flanking sequence information for classification (Quinlan and Hall, 2010). Overlapping intervals were filtered using a dynamic programming technique commonly used in the field of interval scheduling. Intervals were weighted by their G4Hunter score, such that the maximum number of high scoring non-overlapping intervals was yielded.

To ground truth score these sequences, `bedtools map` was used to intersect them with the G4Seq dataset (Chambers et al., 2015), which was downloaded from GEO (GSE63874). Bedgraph files of this data contained percentage mismatch scores for each position of the human genome at 15bp resolution. The G4Seq dataset generated in the presence of potassium was chosen as it was deemed more likely to be of biological relevance than the dataset generated

in the presence of Pyridostatin, a G4-binding drug.

Intervals files and corresponding mismatch scores were read into Python using `pandas` and histograms of log transformed mismatch scores were plotted using `matplotlib` (VanRossum, 1995; Hunter, 2007; Mckinney, 2011). The threshold of approximately 3 for separating G4-forming and non-G4 forming sequences was chosen using `scipy` to determine the local minimum in the histogram (Jones et al., 2001). Joint plots of percentage mismatch score against G4Hunter score were plotted using `seaborn` (Waskom et al., 2014).

Since positive training examples were outweighed by negative ones in this dataset by a factor of 10:1, random under-sampling of negative examples was conducted using `imblanced-learn` to attain a ratio of 2:1 (Lemaître et al., 2001). This filtered dataset was shuffled and written to disk in bed format, and `bedtools getfasta` was used to extract sequences for each interval from hg19 (Quinlan and Hall, 2010). Sequences were then one hot encoded and loaded into HDF5 format for training using `h5py` (Andrew Collette, 2013). For training of models on trinucleotide content, trinucleotide content statistics were extracted and loaded into HDF5 format.

1.2.3 Model Training and Validation

All models were trained in Python using `Keras` with `TensorFlow` backend (Abadi et al., 2016; Chollet, 2018). The trinucleotide Multi-Layer Perceptron model contained three hidden layers with 16 units per layer. These were trained using the ADAM optimiser and binary crossentropy loss function, with a dropout rate of 0.2 on all layers. The convolutional portion of G4Seeqer was made up of two convolutional layers with 8 filters and kernel size of 3 and ReLu activation, followed by a maximum pooling layer with step size of 2. This was connected to a bidirectional Long Short Term Memory layer with 8 units and Tanh activation. The final hidden layer was a fully connected layer with 16 units, ReLu activation

and a dropout rate of 0.5. G4Seeqer was trained using the RMSprop optimiser and binary crossentropy loss function.

All models were trained on 80% of the training data with 10% used for validation. Training was conducted for a maximum of 30 epochs, but with early stopping when the change in validation loss was less than 0.0005 for more than 3 Epochs. The Trinucleotide MLP model converged to this minimum change after 8 Epochs, whilst G4Seeqer converged after 15 Epochs.

Models were validated on 10% of the total data held out for testing purposes. Receiver Operator Characteristic (ROC) and Precision Recall (PR) curves were generated using `scikit-learn` and plotted with `matplotlib` (Hunter, 2007; Pedregosa et al., 2011). ROC/PR curves for G4Hunter are produced using the modified method. For comparison against Quadron, the Quadron source code was downloaded from GitHub and installed (Sahakyan et al., 2017). Since the flanking sequences required for Quadron are longer than those used for G4Seeqer, test set sequences were increased in size by 50bp in each direction to 228bp. Sequences were extracted using `bedtools getfasta` and run through Quadron (Quinlan and Hall, 2010). For intervals which contained multiple Quadron scoring motifs, the highest score was used. For intervals which had no motifs scored by Quadron, a score of zero was assigned.

1.2.4 BG4 Analysis

NarrowPeak BED files of BG4 ChIP-seq peaks were downloaded from GEO accession GSE76688 (Hänsel-Hertsch et al., 2016). To accommodate Quadron's flanking sequence requirements, the size of the BG4 intervals was increased by 50bp in each direction using `awk` (Aho et al., 1988). A BG4-negative peak set was generated using `bedtools shuffle` (Quinlan and Hall, 2010). Shuffling was performed such that an equal number of simliarly sized intervals were selected that excluded gaps in the genome or BG4-positive peaks. Positive

and negative peaks were concatenated and sequences were extracted using `bedtools getfasta` (Quinlan and Hall, 2010). Predictions were made on these sequences using G4Seeqer/G4Hunter/Quadron, and the maximum scoring interval per peak was assigned as the overall score of the peak. Where a model did not make any predictions in a peak, it was assigned a score of zero. Receiver Operator Characteristic (ROC, false positive rate plotted against true positive rate) and Precision Recall (PR, precision plotted against recall) curves were generated using `scikit-learn` and plotted with `matplotlib` (Hunter, 2007; Pedregosa et al., 2011).

1.2.5 rG4seq Training Data Preprocessing

To produce training data for rG4seeqer, G4hunter windows were predicted in hg19 using a window size of 50bp and a threshold of 0.75. These were intersected with human exons using `bedtools` to get a set of 186279 putative RNA G4 forming sequences. RNA G4s identified by rG4seq in the presence of potassium (Kwok et al., 2016) were downloaded from GSE77282 (`GSE77282_K_hits.bed.gz`), and intersected with the G4hunter windows to identify RNA G4 positive and negative examples. Of the 3383 identified RNA G4s in the rG4seq dataset, 2811 (83%) overlapped with G4hunter windows. Since the ratio of negative to positive examples was extremely high, negative examples were undersampled to a ratio of 2:1 using `imbalanced-learn` (Lemaître et al., 2001). These were then shuffled and written to disk in bed format, and `bedtools getfasta` was used to extract sequences for each interval from hg19 (Quinlan and Hall, 2010). Sequences were then one hot encoded and loaded into HDF5 format for training using `h5py` (Andrew Collette, 2013).

1.2.6 rG4seq Transfer Learning

Model weights trained on the G4Seq dataset were reloaded in the same architecture for training on the rG4seq dataset, using `Keras` and `tensorflow` (Abadi et al., 2016; Chollet, 2018). Weights from the initial convolutional layers were fixed (i.e. made untrainable), and only LSTM weights and final dense weights were trained. The model was trained on 6014 samples (80%), validated on 752 samples (10%), and tested on 752 samples (10%). Training was conducted as for G4Seeqer, but for a maximum of 200 epochs, with early stopping after 25 epochs if validation loss did not improve. Initial learning rate was set at 0.001 but reduced by a factor of 1/3 after 15 epochs when no reduction in validation loss was seen.

1.2.7 Comparison to G4RNA Screener

Supplementary data containing 368 RNA sequences, their experimentally determined G4 forming status, and their predicted score from G4RNA Screener were downloaded (Garant et al., 2017). Sequences greater than 128bp were filtered to give a total of 347 sequences. Average rG4Seeqer and G4Seeqer predictions for these sequences were made by generating 1000 randomly paddings for each sequence, one hot encoding, and performing forward pass through the network. G4RNA Screener scores used in the ROC curve were taken directly from the supplemental material (Garant et al., 2017).

1.2.8 Mutation Mapping analysis

Mutation mapping was applied to human promoter regions from the ENSEMBL regulatory build (Zerbino et al., 2015), which was originally generated using `ChromHMM` (Ernst and Kellis, 2017). Promoter sequences were extracted from hg38. Mutation mapping was implemented as in Alipanahi et al. 2015: candidate

sequences were edited at each position to each nucleotide, and the resulting sequences were scored for G4 formation using G4Seeqer (Alipanahi et al., 2015). Heatmaps were generated in Python using `seaborn` (Waskom et al., 2014).

1.2.9 G-Triplex and Hairpin analyses

G-triplex motifs were predicted in the hg19 genome using an in house script, using the pattern $G_X N_{1-4} G_X N_{1-4} G_X$ where $3 \leqslant X \leqslant 6$. Candidate triplexes which overlapped with or were contained within Quadparser motifs (pattern $G_X N_{1-7} G_X N_{1-7} G_X N_{1-7} G_X$ where $3 \leqslant X \leqslant 6$) were subtracted to produce only triplex motifs which could not form "classical" G4s. To assign mismatch scores to these sequences, they were increased in size by 50bp in each direction using `bedtools slop` (Quinlan and Hall, 2010), and mismatch scores from the G4Seq dataset were mapped using `bedtools map`. Distances to next G-run were measured using Python scripts.

G-hairpin motifs were predicted in the hg19 genome using the same script, and the pattern $G_X N_{1-4} G_X$ where $4 \leqslant X \leqslant 6$. Candidate hairpins which overlapped with or were contained within Quadparser motifs or G-triplex motifs were filtered. Intervals were increased in size by 50bp in each direction using `bedtools slop` (Quinlan and Hall, 2010), and mismatch scores from the G4Seq dataset were mapped using `bedtools map`. Distances to next G-hairpin were measured using `bedtools closest`. Triplex and hairpin histograms and boxplots were generated in Python using `matplotlib` and `seaborn` (Hunter, 2007; Waskom et al., 2014).

1.2.10 Median model score experiments

For G4Seeqer scoring of experimentally validated G4s from Guédin et al 2010., sequences were recreated from the information in Table 1, and padded using randomly generated sequences to 128bp in length (Guédin et al., 2010). Left and right padding lengths were also varied at random. 1000 randomly padded sequences were generated and scored per input sequence. Scatter plots of UV melting temperature vs. median G4Seeqer score were produced using `matplotlib` (Hunter, 2007). Errorbars are 68% confidence intervals.

Synthetic sequences used for loop length and G-register experiments were 3 tetrad Quadparser conforming sequences. Each sample contained 5000 randomly generated sequences. Left and right padding sizes and nucleotide contents were varied within random samples. Loop nucleotide contents were also varied. For G-register experiments, the extra G per run was randomly assigned to either the left of right side of the G-run. Loop lengths of 3 were used. Line plots for loop length experiments were generated using `matplotlib`, and errors are 68% confidence intervals (Hunter, 2007). Boxplots were generated using `seaborn` (Waskom et al., 2014).

1.2.11 G-register Experiments

For G-register mismatch experiments, 3 tetrad Quadparser conforming G4s from the hg19 genome were identified and the corresponding mismatch score was extracted using `bedtools map` (Quinlan and Hall, 2010). The number of tetrads with G-register was counted. Boxplots were generated using `seaborn` (Waskom et al., 2014).

1.2.12 Human and Mouse G4 Subpopulation Analyses

Example Human and Mouse G4 populations were predicted in hg19 and mm10 using the Quadparser pattern with loop lengths of 3. All possible G4s conforming to this pattern were generated using python `itertools`. Venn diagrams were generated using `matplotlib_venn`. P-values were produced using hypergeometric tests. Dinucleotide complexity was defined as the total number of unique dinucleotides contained in the motif. Histograms and kernel density estimate plots were produced using `seaborn` (Waskom et al., 2014). For visualisation of PG4 distributions, a sample of 50000 motifs were randomly selected from the total population of all possible Quadparser motifs with loop lengths of three. These were transformed into two components using UMAP dimensionality reduction, with Hamming distance as the distance metric (McInnes and Healy, 2018). Sequences which appear in the human and mouse genome were extracted from the sample. 2D Kernel Density Estimate plots for the full sample, and hg19 and mm10 subsets, were generated using `seaborn`.

1.3 Results and Discussion

1.3.1 Candidate G4 proposal

One major drawback to any machine learning method for G4 prediction over existing regular expression or pattern based methods is the relative expense of computation. It is therefore not sensible to train and classify a neural network model on all possible input sequences from a genome. Instead we decided to use an existing method, the G4hunter algorithm proposed by Bedrat et al. (Bedrat et al., 2016), to produce candidate regions which could then be labelled as true positive G4s or non-G4s, and used as input for model training.

We reimplemented the G4hunter algorithm with some minor modifications (Bedrat et al., 2016). Bedrat et al's method run length encodes the sequence of interest and scored G-runs as the square of the run length, and C-runs as the negative of the square of the run length. These scores are then summed to give an overall score for the sequence. This method was chosen as it was assumed, based on earlier work, that a high G-content on both strands would make the G-Quadruplex unfavourable compared to double stranded DNA. Since we wished to make as few assumptions as possible, and given that the G4seq dataset stems from G4 formation in *in vitro* single stranded DNA, we altered the method to produce two scores. G runs score positively on the positive strand and C runs score positively on the negative strand. This means the sequence d(GGGCCC) would yield a high score on both strands rather than a single score of zero.

To produce candidate regions for model training, we ran the modified G4hunter method on the human genome (hg19) using a window size of 50bp, a step size of 5bp and a threshold of 0.75. Unstringent values were chosen to produce a high recall, i.e. capture as many true positive G4 structures as possible. These settings produced intervals which overlapped with all PG4 sequences predicted

by the Quadparser method using maximum loop lengths of 12. It also produced significantly more sequences that did not conform to the Quadparser method, some of which are likely to form G4 structures.

1.3.2 Training Data preprocessing

To created training sequences, the 50bp candidate regions from the G4hunter method were increased in size by 39bp in each direction to produce intervals of 128bp in length, since previous work has suggested that flanking regions are an important determinant of G4 stability. Clusters of overlapping intervals were filtered to produce only the interval with the highest G4Hunter score (in cases of ties, a random highest scoring interval was selected). This produced a total of 6,237,943 candidate intervals. Each candidate interval was then scored by mapping the value of the maximum scoring overlapping window from the G4seq dataset (Chambers et al., 2015), which contains percentage mismatches (%mm) from sequencing in the presence of potassium vs. absence of potassium, in 12bp windows. Regions of high %mm on the positive strand indicate a G4 structure on the negative strand, and vice versa.

Plotting the distribution of the log of the %mm scores produced a bimodal distribution with a peak around 1 (corresponding to 2-3% mismatch) and another around 3.5 (corresponding to approximately 30% mismatch) (Fig. 1.1a). We determined the local minimum in the histogram between the two peaks to be approximately 3 (around 20% mismatch), therefore we used this value to split the data into G4 positive and G4 negative subsets. This yielded more than 10 times as many G4 negative sequences than positive, however (5,809,719 negative to 428,224 positive). Since maintaining such an imbalance in the training data would produce a poor classifier, we undersampled the G4 negative class to a ratio of 2:1, yielding a total of 1,284,672 sequences.

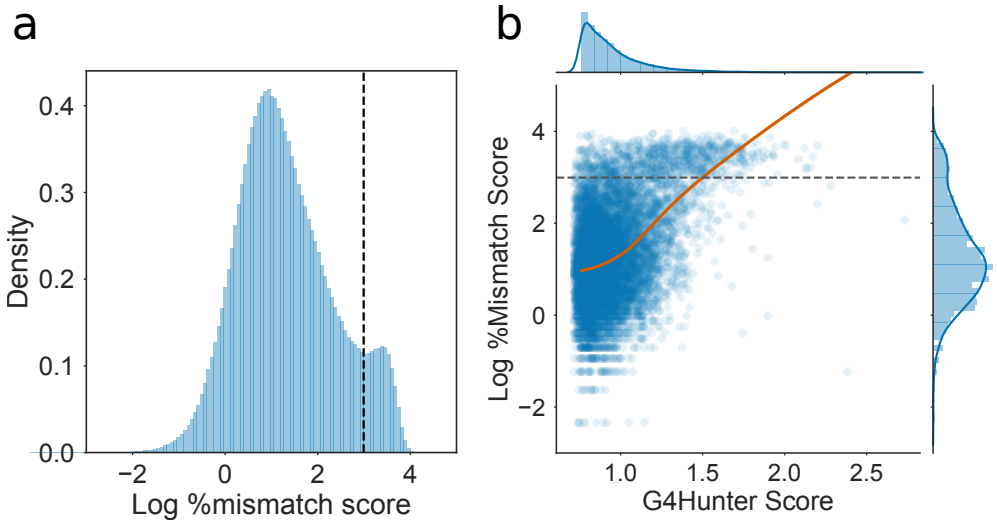

Figure 1.1: Mismatch scores of candidate sequences identified by G4Hunter: a) Histogram of log percentage mismatch score from the G4Seq dataset, for the 50bp sequences identified by G4Hunter (threshold of 0.75). Dashed line shows the threshold chosen to delimit G4 positive and G4 negative sequences. This corresponded to around 20% mismatch score. **b)** Joint plot of log mismatch score against G4Hunter score for 10000 randomly sampled sequences. Orange line shows lowess curve fit.

1.3.3 Model selection and training

Previously published G quadruplex prediction methods which utilise machine learning techniques have used derived features such as trinucleotide content to feed to models (Garant et al., 2017; Sahakyan et al., 2017). These features result in the loss of some spatial information about the sequence, however. For example, the sequence `GGTGGTGGTGGGGGG` has the same trinucleotide composition as `GGGTGGGTGGGTGGG`, but is unlikely to have equivalent G4 forming propensity. Furthermore, Quadron derived features require input sequences to conform to the QuadParser regular expression (Huppert and Balasubramanian, 2005), meaning that Quadron is only able to improve the precision of the QuadParser method, and not the recall. We opted for a neural network involving convolutional layers (those often used for image classification) that could make predictions directly from the sequence itself, without any derived features whatsoever. This allows us to make no assumptions about potential G4 patterns in the dataset. The overall architecture selected was a convolutional-recurrent neural network (Fig. 1.2), which has previously been used to identify regulatory motifs in DNA (Quang and Xie, 2016). The architecture consists of two one dimensional convolutional layers with a kernel size of 3, to capture local features in the sequence. A maximum pooling layer then reduces the size of the feature space by half. These features are then fed to a bidirectional Long Short Term Memory (LSTM) layer, which is able to learn and recognise long distance relationships between features in the sequence. The model outputs a single value between zero and one of the probability of G4 formation. The dataset was split into three for training and testing: 1332565 sequences (~80%) were used for training, 166571 (~10%) for in-training model validation, and 166576 (~10%) for post-training model testing.

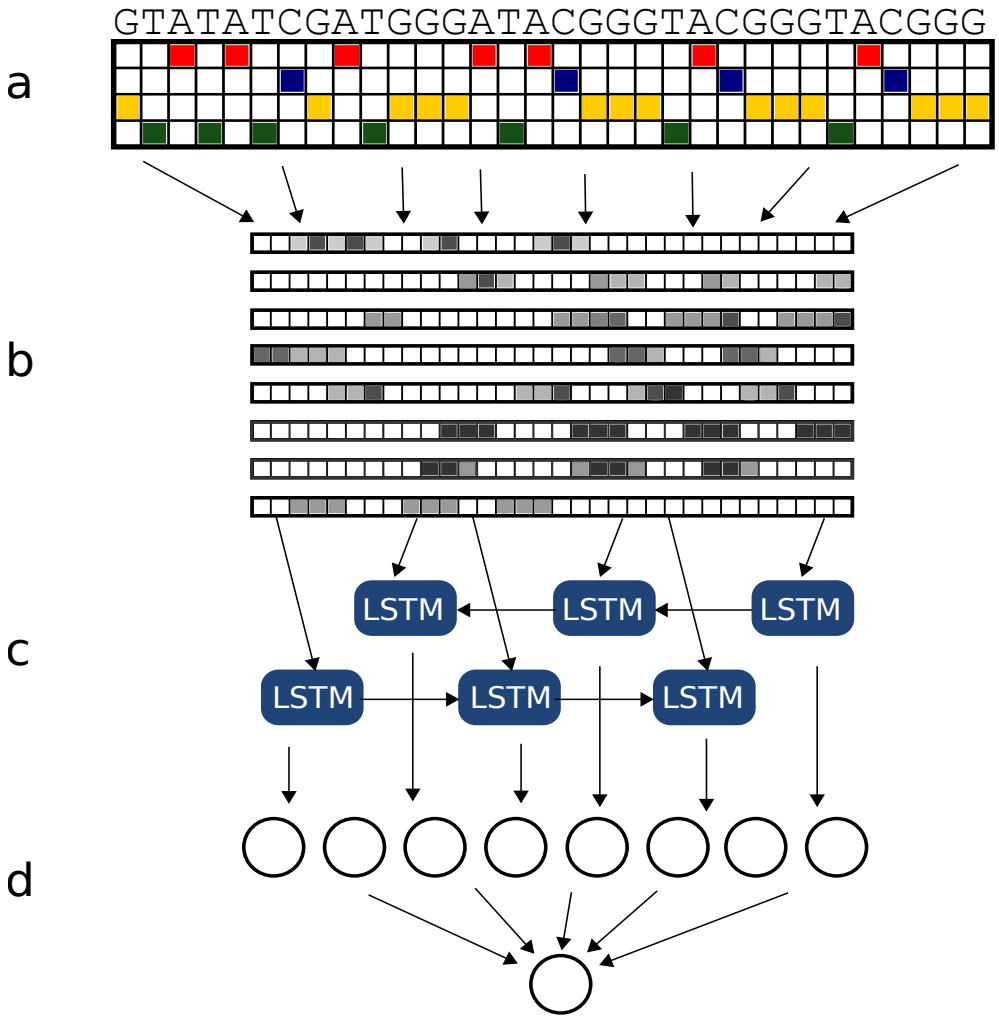

Figure 1.2: G4Seeqer architecture Adapted from Quang & Xie 2016. **a)** Sequences are one hot encoded to produce a matrix which can be processed by the neural network. **b)** Input matrices are passed through a convolutional layer. Each layer contains 8 filters which are trained to recognise local patterns on the scale of 3-6bp in size. **c)** Convolutional features are passed through a bidirectional Long Short Term Memory (LSTM) layer. This layer recognises long distance interactions between features which might combine to produce G4s. **d)** Finally, features are passed through a fully connected layer. Output from the model is a single probability of whether the sequence forms a G4.

1.3.4 Comparison to existing methods

We benchmarked our technique (hereafter referred to as G4Seeqer) using the 10% of the data reserved for testing. The model was compared to our modified G4Hunter method, as well as a multi-layer perceptron model trained on trinucleotide frequencies derived from the same dataset as was used to train G4Seeqer. This model allows us to compare the methodology of G4RNA Screener (Garant et al., 2017) to our own method, since G4RNA Screener was originally trained on a database of RNA G Quadruplexes, and may not perform as well on a dataset derived from DNA. The performance of the methods were calculated using the Receiver Operating Characteristic area under curve (ROC AUC). We found that neither G4Hunter (Bedrat et al., 2016) nor the G4RNA-like method performed as well as G4Seeqer on the test dataset (AUC G4Hunter 0.82, G4RNA Screener-like 0.90, G4Seeqer 0.94) (Fig. 1.3a-b). This is likely due to the loss of sequence spatial information in the former methods.

To benchmark our method against the other G4Seq trained machine learning G4 prediction package, Quadron, we downloaded and installed the Quadron source code (Sahakyan et al., 2017). Quadron was used to score sequences from the held out test set and compared to the performance of other methods. Quadron requires larger flanking regions of 50bp, so test set intervals and sequences were increased in length by 50bp in both directions to produce test sequences of length 228. This was deemed the best way to compare the two methods, however was still not ideal since Quadron may have been trained on some or all of the test sequences. Regions which had predictions associated with them by G4Hunter and G4Seeqer but had no associated predictions from Quadron (due to not conforming to the Quadparser regular expression) were given a Quadron score of zero. The ROC curve of Quadron (AUC 0.7) ((Fig. 1.3a-b)) had interesting properties: the model is capable of producing a true positive rate of around 30% with a false positive rate of less than 1%, however

shortly beyond this point in the ROC curve the curve becomes linear. This is because Quadron is only capable of scoring sequences which conform to the Quadparser method, with all other sequences being scored zero. Since these sequences only account for a 25% of all the potential PG4 forming sequences in the test set, the model does not perform well on the full dataset. Because it makes no assumptions about the input sequence, G4Seeqer is able to make accurate predictions for all forms of PG4 in the test data, as reflected in the ROC curve.

We were interested in how G4Seeqer compared against Quadron on only sequences which conformed to the Quadparser motif. We therefore filtered our dataset for intervals on which Quadron had made a prediction, and replotted the ROC curves and Precision Recall curves for the filtered data (Fig. 1.3c-d). As expected the AUC for Quadron was much better on this filtered set (AUC 0.93), however it was still outperformed by G4Seeqer (AUC 0.94), suggesting that G4Seeqer captures the same or more explanatory information directly from the input sequence.

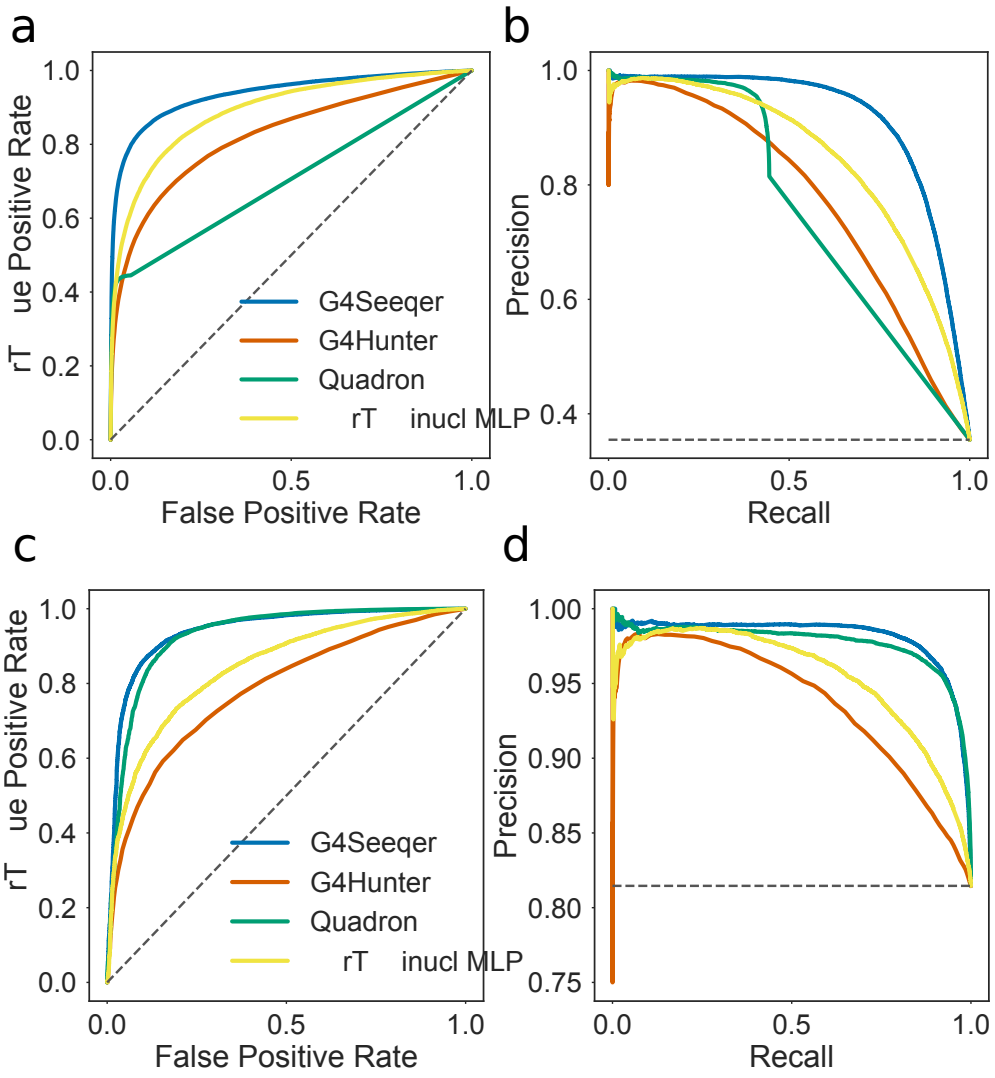

Figure 1.3: Validation curves for G4Seeqer method a) Receiver Operator Characteristic (ROC) curve showing the performance of G4Seeqer, Multi-Layer Perceptron (MLP) trained on trinucleotide contents, Quadron, a Gradient Boosted Machine model (Sahakyan et al. 2017) and the G4Hunter method (Bedrat et al. 2016), on a held out test set of the G4Seq dataset (10% of total dataset). **b)** Precision-recall curves showing the performance of G4Seeqer, trinucleotide MLP, Quadron, and G4Hunter on the same dataset. **c)** ROC curve and **d)** Precision Recall curve showing the performance on sequences from the test set conforming to the Quadparser motif.

1.3.5 BG4 ChIP-seq data evaluation

Further model validation was performed on G4s experimentally validated by an entirely different technique, namely G4-chromatin immunoprecipitation (BG4) (Hänsel-Hertsch et al., 2016). The BG4 dataset is arguably more biologically relevant than G4seq since G4 structures are not induced by addition of potassium, and are captured from native chromatin. BG4 peak intervals were shuffled to produce a set of G4 negative sequences, and then the performance of the models was evaluated on the real and shuffled peaks. For each BG4 interval, the highest scoring overlapping prediction for each model was assigned. Any intervals with no overlapping predictions were scored zero for that model. We tested the G4Seeqer, G4Hunter and Quadron methods. As with the G4Seq test dataset, we found that Quadron performed reasonably for Quadparser conforming BG4 peaks, but was unable to identify most of the true positive BG4 peaks due to its restriction to the pattern. G4Seeqer performed better on all BG4 peaks, with an AUC of 0.71, however was only marginally better than the G4Hunter technique (AUC 0.7) (Fig. 1.4). These results suggest that the information within the G4Seq dataset, when captured by a suitable model, is predictive of G4s in an *in vivo* setting.

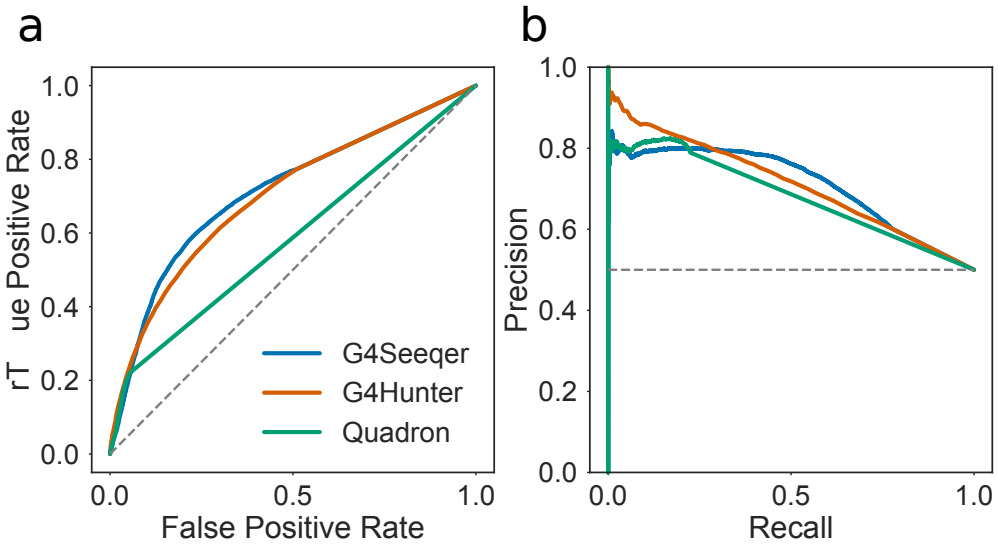

Figure 1.4: Detection by BG4 peak sequences using G4Seeqer a) Receiver Operator Characteristic (ROC) curve showing the performance of G4Seeqer, Quadron, and the G4Hunter method, on BG4 ChIP-seq peaks and randomly shuffled negative sequences.

1.3.6 Transfer learning on RNA G4Seq (rG4seq) Dataset

PG4s with the same sequence are likely to have slightly different G4 forming potentials in DNA and RNA, due to the chemical differences in these molecules. The sugars which make up the backbone of RNA are riboses, which have an extra 2' hydroxyl group compared to the deoxyribose found in DNA. This extra hydroxyl group is thought to have a number of implications for G4 formation: it increases the number of backbone hydrogen bonds in the G4, increasing its enthalpic favourability and its entropic favourability (by reducing the number of coordinated water molecules) (Collie et al., 2010). Furthermore, the 2' hydroxyl introduces steric constraints which make parallel RNA G4s much more favourable than anti-parallel ones (Collie et al., 2010). Given these differences, it is likely therefore that a model specifically trained on DNA G4 sequences will not perform optimally on RNA G4s.

To address this issue, we decided to retrain G4Seeqer using the rG4seq dataset produced by Kwok et al. 2016, to create an rG4Seeqer model (Kwok et al., 2016). Data was prepared similarly to the data for G4Seeqer: candidate regions were selected from human exonic sequences using G4hunter with window size of 50bp and a threshold of 0.75. This yielded a set of 186279 putative RNA G4 forming sequences, which were increased by 39bp in each direction to get flanking sequences. These were then intersected with rG4seq hits collected under potassium stabilising conditions (Kwok et al. 2016). Of the 3383 identified RNA G4s in the rG4seq dataset, 2811 (83%) overlapped with G4hunter windows. rG4seq negative examples were undersampled with a ratio of 2:1 to yield 7518 training samples. 80% of these were used for training, with 10% for validation and 10% held out for testing.

Because of the significantly smaller size of the rG4seq derived training set, we found that the method for training which yielded most optimal results was transfer learning from the G4Seeqer model. Weights of the initial convolutional

feature extraction layers were therefore held constant, and only the weights of the LSTM layers (which find long range interactions) and dense output layers were retrained.

Testing was first conducted on the held out set of 752 sequences using G4Hunter, G4Seeqer and the newly trained rG4Seeqer. G4Seeqer significantly outperformed G4Hunter (AUC 0.9 vs 0.83 respectively), suggesting that the information extracted from the G4Seq dataset is applicable to the rG4seq dataset (Figure 1.6). rG4Seeqer outperformed both methods, however (AUC 0.95), demonstrating that domain specific information is better for predicting RNA G4s in the rG4seq dataset (Figure 1.6).

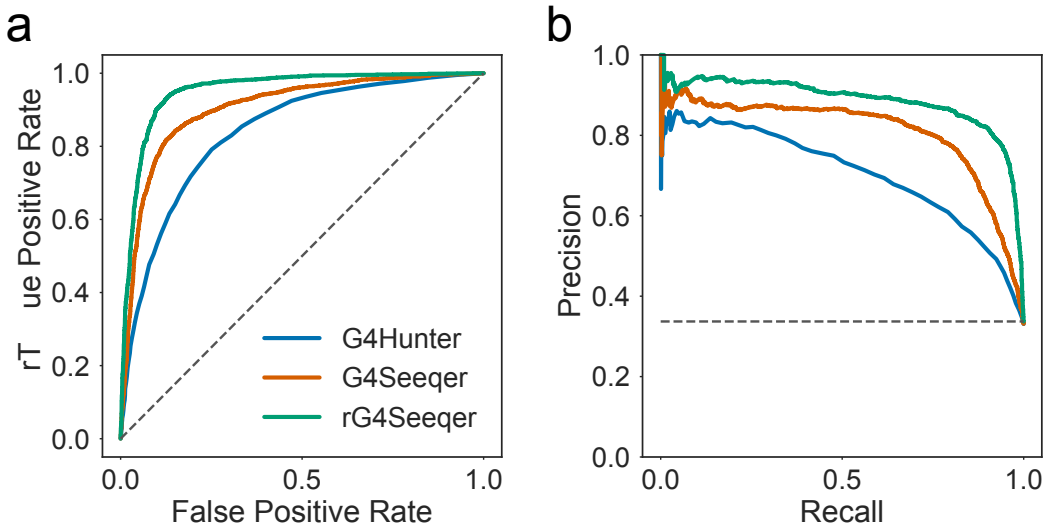

Figure 1.5: Validation curves for rG4Seeqer method a) Receiver Operator Characteristic (ROC) curves showing the performance of rG4Seeqer, G4Seeqer and the G4Hunter method (Bedrat et al. 2016), on a held out test set of the rG4Seq dataset (10% of total dataset). **b)** Precision-recall curves showing the performance of rG4Seeqer, G4Seeqer, and G4Hunter on the same dataset.

We sought to test rG4Seeqer on G4s identified by a variety of physical methods, using the set of RNA G4s curated by Garant et al. for their model, G4RNA Screener (Garant et al., 2017). We used 347 sequences from this dataset, of which 169 sequences were G4 positive and 178 were G4 negative. G4Seeqer and rG4Seeqer predictions were calculated by padding with random sequences to a length of 128bp before one hot encoding. This was conducted 1000 times for each sequence and the mean score was taken. G4RNA screener scores were taken directly from the supplemental information of Garant et al. 2017. G4RNA screener was found to perform best on the dataset (AUC 0.91), perhaps unsurprisingly since it was trained directly on the sequences. Perhaps more importantly, rG4Seeqer significantly outperformed G4Seeqer on the dataset (AUC 0.89 vs 0.82), showing that the rG4seq trained model generalises better to RNA G4 sequences than the G4seq trained model.

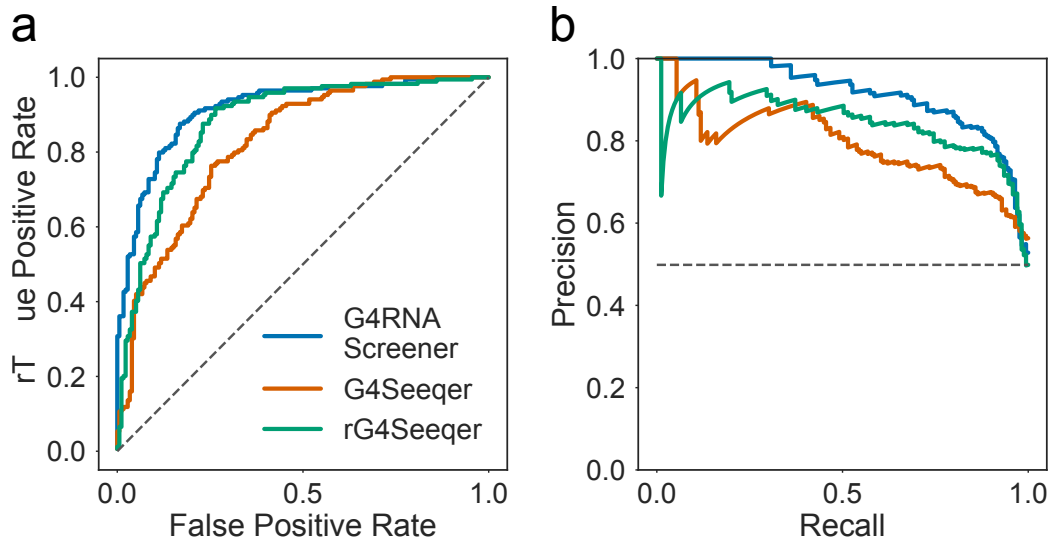

Figure 1.6: Validation of rG4Seeqer on *in vitro* experimentally categorised RNA sequences a) Receiver Operator Characteristic (ROC) curves showing the performance of rG4Seeqer, G4Seeqer and the G4RNA Screener method (Garant et al. 2017), on the G4RNA dataset curated by Garant et al. **b)** Precision-recall curves showing the performance of rG4Seeqer, G4Seeqer, and G4RNA Screener on the same dataset.

1.3.7 Interpreting G4Seeqer output using Mutation Mapping

One common complaint about neural network techniques is that the complexity of the models they produce make them “black boxes” which are impossible for humans to understand or extract useful knowledge or rules from. It is possible, however, to visualise some of the output of a neural network through various means. One commonly used method for interpretation is the “saliency” of the network, which can produce heatmaps showing the attention of the network to specific regions of the input image or sequence. This can be used to determine the important aspects of the input in classification. For biological sequences, previous studies have used similar methods, called “Mutation Maps”, to analyse the importance of individual nucleotide positions on convolutional neural network model predictions (Alipanahi et al., 2015; Quang and Xie, 2016). Simply, the importance of each particular nucleotide is evaluated by replacing it with each of the other three bases, and calculating the change in model score. This is then used to build a heatmap which can be used to visualise the importance of each position.

Previous studies have highlighted a possible role for G4 forming sequences in promoters, with G4 formation tending to have a positive effect on expression, particularly in proto-oncogenes (Eddy and Maizels, 2006; Hänsel-Hertsch et al., 2016). We therefore decided to use the mutation map approach to characterise PG4s in promoter regions extracted from the ENSEMBL regulatory build (Zerbino et al., 2015). Promoter sequences were screened using G4Seeqer and single base mutation maps were created for each candidate PG4, including those regions where the neural network score was low. Unsurprisingly, all of the most deleterious single base substitutions (causing a score reduction of more than 0.9) predicted by G4Seeqer mutation mapping were G->H changes.

We identified PG4 sequences scoring more than 0.9 for which a single G->H change resulted in a reduction of as much as 0.9 in score (i.e. switched the

score from strongly PG4 positive to strongly PG4 negative) (Fig. 1.7a). Analysis of the mutation maps for these sequences showed that the majority of contained regions containing three G-runs with short connecting loops. These tended to have a long final loop to the next homopolymeric G-run, or no final G-run within the window size. Any G->H mutation in these G-dense regions strongly affected the predicted G4 forming ability. Recent work by Hou et al. has shown that formation of G4 structures in human telomeric sequences occurs via a stable G-triplex structure (Hou et al., 2017). We believe these results suggest that the G4seq dataset is either capturing mismatches caused by G-triplex structures, or by G-quadruplexes formed from short range G-triplex interaction with more long range single G-runs. To further illustrate this we predicted all short looped (1-4bp) G-triplexes in the human genome which did not overlap with a Quadparser predicted PG4 (loop lengths 1-12bp). We found that 35% of these were associated with a %mm score greater than 20%, suggesting the formation of some secondary structure (Fig. 1.7b). To see if medium range G-run interactions might stabilise these, we then measured, for each G-triplex, the distance to the next run of at least three Gs on the same strand, and compared this to the %mm score. We found a weak negative correlation between distance and %mm score (Spearmans rho -0.2) for G triplexes with a G-run less than 100bp away, suggesting that these longer range interactions do occur but become weaker with distance (Fig. 1.7c). This could be due to a reduction in G4 stability with loop length, but could equally be explained by a reduction in the likelihood of the next G-run being contained in the same sequenced fragment. No clear difference was observed in the correlation of %mm score with upstream or downstream G-runs (Spearmans rho -0.17 and -0.15 respectively) suggesting there is no preference for the long loop region to be at the 5' or 3' of the G triplex.

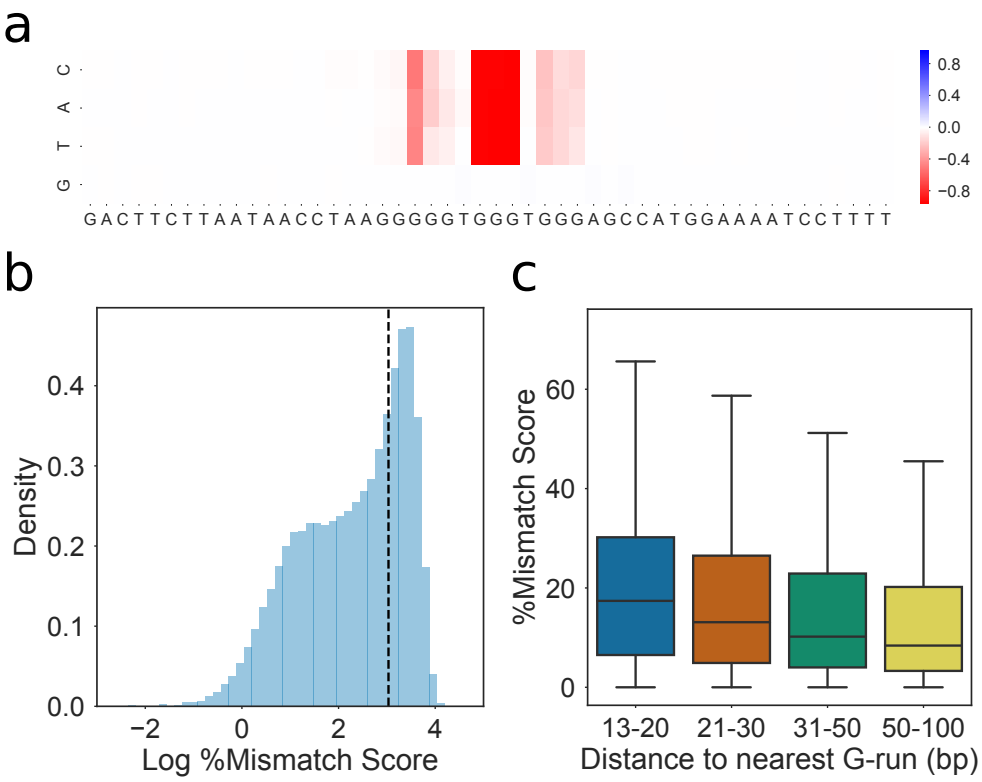

Figure 1.7: Identification of G-triplex structures by G4Seeqer Mutation Maps a) Mutation map showing the a high scoring (0.99) G4Seeqer motif which may form a G-triplex. Mutation of any base in the central G-run of the motif is sufficient to reduce the score by up to 80%. **b)** Histogram of log percentage mismatch score for motifs conforming to a G-triplex like pattern. The bimodal distribution suggests that many of these motifs form structures which disrupt polymerase in the presence of potassium. **c)** Boxplot showing the relationship between %mm score and distance to next G-run in G-triplex structures. The negative correlation suggests G-triplexes might recruit distant G-runs to form G4s.

We also noted that G4Seeqer was positively labelling certain sequences which contained only two G-runs, usually of greater than 4 bases in length (Fig. 1.8a). These scores were also very sensitive to G->H mutations. Based on the folding dynamics work by Hou et al, we hypothesised that these sequences might form G-hairpins, which could associate with other nearby hairpins to form G4 structures (Hou et al., 2017). To test this, G-hairpins with G-run lengths greater than four were predicted and filtered to remove any overlaps with predicted QuadParser G4s (loop length 1-12) or G-triplexes (loop length 1-4). 27% of these sequences had %mm scores greater than 20% (Fig. 1.8b). For each predicted G-hairpin, the distance to the nearest G-hairpin was calculated and correlated with %mm. Again, distance was found to correlate negatively with %mm score (Spearmans rho -0.18), suggesting that these hairpins may associate with each other to form G4s (Fig. 1.8c).

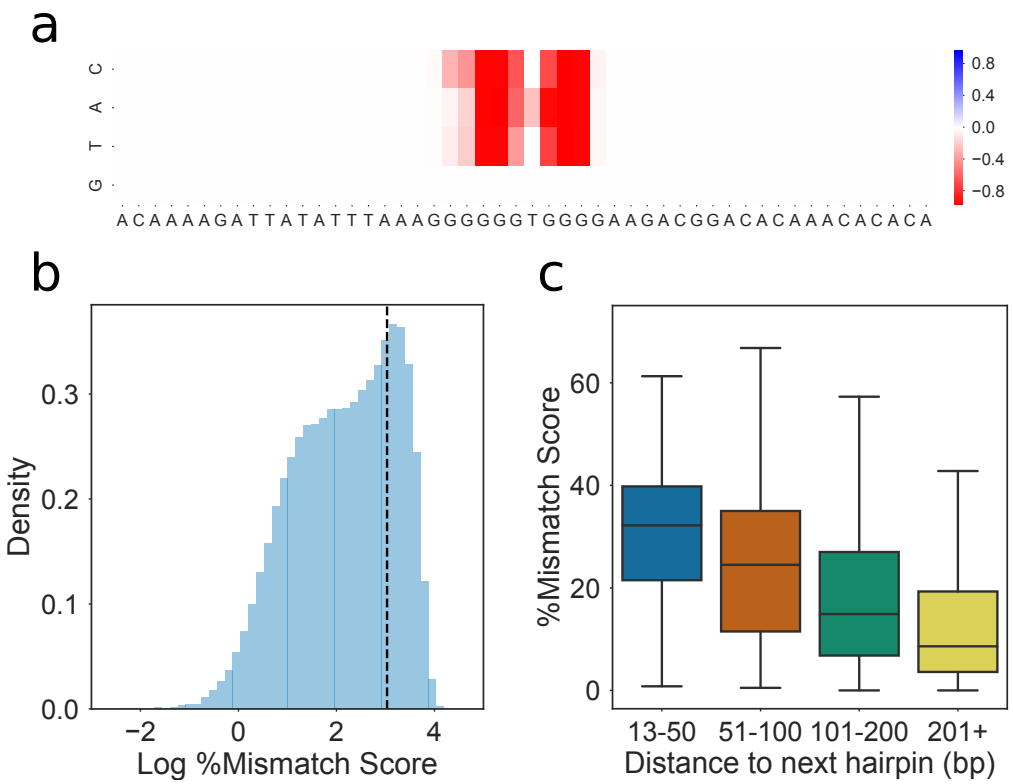

Figure 1.8: Identification of G-hairpin structures by G4Seeqer Mutation Maps a) Mutation map showing the a high scoring (0.99) G4Seeqer motif which may form a G-hairpin. Mutation of any base in the core motif is sufficient to reduce the score by up to 80%. **b)** Histogram of log percentage mismatch score for motifs conforming to a G-hairpin like pattern. The bimodal distribution suggests that many of these motifs form structures which disrupt polymerase in the presence of potassium. **c)** Boxplot showing the relationship between %mm score and distance to next G-hairpin for G-hairpin structures. The negative correlation suggests G-hairpins might interact with other relatively distant hairpins to form G4s.

In order to study how interactions between pairs mutations might affect predicted G4 stability, we developed a pairwise mutation map, in which pairs of Gs in each sequence were combinatorially mutated to Ts. We then analysed the resultant mutation maps to identify pairs of G->T transversions which interact to reduce predicted G4 forming potential more strongly than each individual mutation. Perhaps unsurprisingly, we found that in sequences which had more than four G-runs, or had G4s containing features which might form G-triplexes or G-hairpins, mutations which disrupted peripheral G-runs did not have a strong effect on predicted stability. Combinations of mutations which disrupt multiple G-runs had a much stronger effect on stability, however.

1.3.8 Loop Length and G4 stability

To determine whether G4Seeqer probability scores supported previous work on the relationship between G4 loop length and stability, we first downloaded UV melting temperatures for three tetrad G4 sequences from Table 1 of Guédin et al. 2010. The majority of these G4 sequences contain only runs of Gs and Ts. In each experiment, one or two of the three T loops were held at a constant length of either 1 or 3, and the other loops varied from 1 up to 15 bp in length (Guédin et al., 2010). For each sequence, we produced 1000 sequences padded to 128bp (the input length of G4Seeqer) using randomly generated bases, and used G4Seeqer to predict the stability. We found a very strong correlation (Spearmans rho 0.93, p = 1.1e-35) between empirically determined melting temperature in potassium, and G4Seeqer score (Fig. 1.9), suggesting that G4Seeqer is successfully capturing information about G4 structure which is transferable between conditions. The midpoint of the curve appears to suggest that a melting temperature of around 65 degrees Celsius is required for significant mismatches to occur in the G4Seq dataset. We also noted that G4seeqer output was more variable for sequences with lower melting

temperatures, suggesting that sequence context may be more important for these G4s.

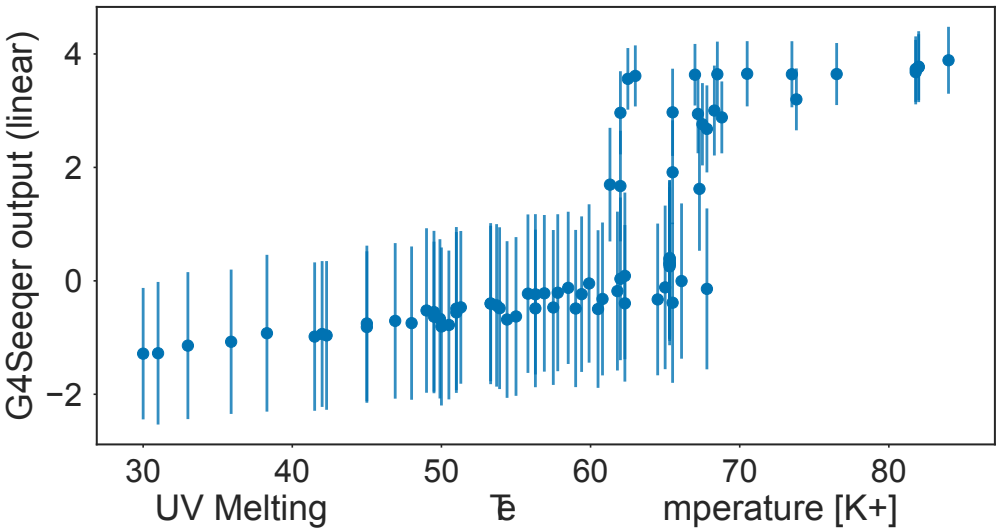

Figure 1.9: G4Seeqer scores correlate with experimentally determined melting temperatures Scatter plot showing median G4Seeqer scores vs. UV melting temperature for sequences from Guédin et al. 2010. Error bars are 68% confidence intervals generated from 1000 iterations of prediction with randomly generated flanking sequences.

We next performed a similar *in silico* experiment to that of Guédin et al., whereby we generated random QuadParser conforming G4 sequences with two loop lengths held at 1bp and the third varied from 1-60bp. Loop regions were constructed by randomly selecting from A C or T. The G4Seeqer score for these sequences was then generated. Unsurprisingly, we found that G4Seeqer score reduced with increasing loop length. This effect was strongest for the central loop of the G4, presumably because varying this loop has a greater effect on the ability to form stable G-triplex intermediates (Fig. 1.10a). We then set the length of the non-varying loops to 3bp and re-ran the analysis for third loop length 1-60bp. For these PG4s, we found that the probability of G4 prediction was much more sensitive to longer loop lengths (Fig 1.10c). There was also no longer a strong difference in prediction when varying the central loop, compared to either loops 1 or 3, possibly suggesting that triplex formation in these sequences is less common.

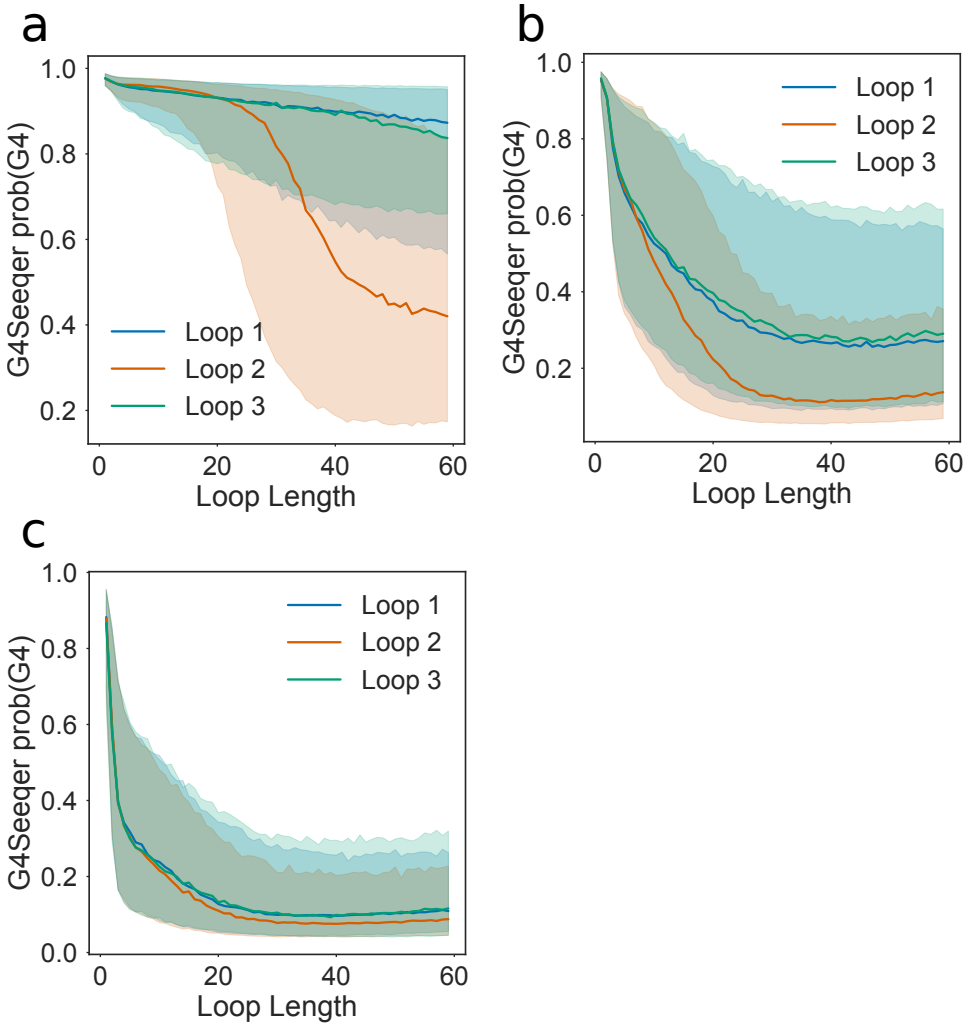

Figure 1.10: Effect of increasing loop length on G4Seeqer score a) Effect of loop length on predicted G4 stability when other loops are held at a constant length of **a)** 1bp, **b)** 2bp or **c)** 3bp. Median value and 68% confidence intervals are produced using 5000 randomly generated sequences (including flanking regions and loop contents) for each pattern analysed.

1.3.9 Effect of G-register on G4 stability

Work by Harkness and Mittermaier has indicated that extra Guanines in some G-runs of a G4 forming sequence might increase the G4 forming potential of the sequence, by allowing exchange between different G4 conformations (Harkness and Mittermaier, 2016). They termed this effect G-register. We analysed the G4seq dataset and the G4Seeqer model outputs to determine whether there was evidence of a relationship between stability and G-register. Firstly, for all PG4s in the human genome conforming to the three tetrad Quadparser motif, we counted the number of G-runs which contained four Gs rather than three. These motifs were then intersected with the G4seq dataset to identify the %mm score. We found that G4seq motifs with greater G-register indeed tend to have higher mismatch scores (Fig. 1.11b). To test whether G4seeqer had successfully identified this pattern, we then randomly generated three tetrad PG4 sequences with loop lengths of 3bp, and introduced addition guanines to increase the G-register. Higher G-register strongly increased the G4Seeqer score of the sequence, showing that the model has successfully learned this feature of G4s (Fig. 1.11a).

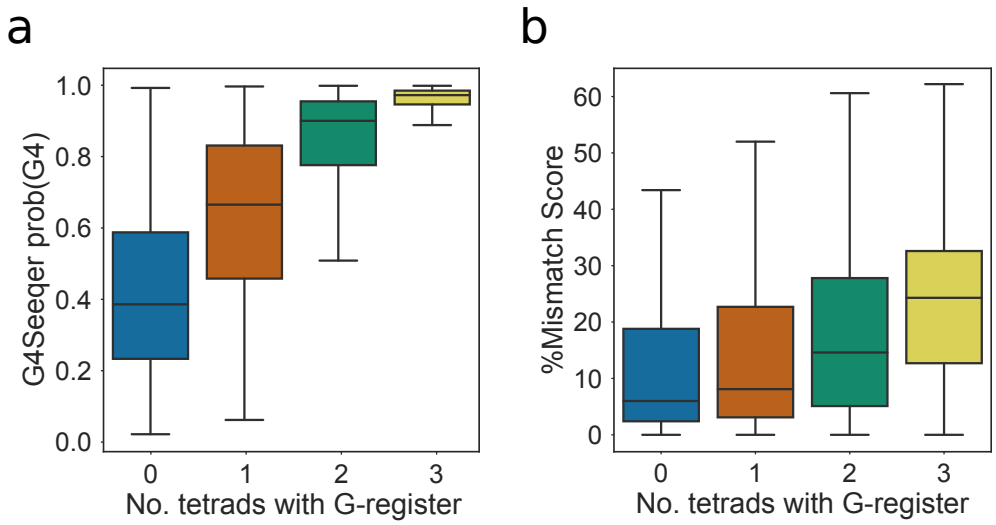

Figure 1.11: Scoring of G-register effects by G4Seeqer a) Boxplot showing G4Seeqer scores for randomly generated 3 tetrad Quadparser conforming sequences with zero to three additional Guanines per run, referred to by Harkness and Mittermaier as G-register. **b)** Boxplot showing relationship between G-register and %mm score in the G4Seq dataset for Quadparser conforming G4s.

1.3.10 Applicability of the model to other genomes

Whilst the Human genome contains a large number of G4 forming sequences, this is not even close to saturating the population of all potential G4s. Indeed, simply considering the Quadparser motif with loop lengths up to 12, there are 1.1×10^{22} different conforming sequences, many orders of magnitude more than there are bases in the human genome. It is probable that the human genome contains a biased subpopulation of all G4s. This might mean that the G4Seeqer model does not generalise well to other genomes which contain different subpopulations. As an example, we analysed 3 tetrad Quadparser motifs from hg19, where all three loops were of length 3. There are 4^9 (262,144) possible sequences fitting this description, of which only 1.4% (3,748) appear in hg19 (Fig. 1.12a). The motifs that did appear tended to be those with lower complexity, as measured by the number of distinct dinucleotides in the sequence, than the total possible population (Fig. 1.12b). The PG4 space of the *M. musculus* genome was also measured, and found to contain 1.7% (4330) of all possible 3 tetrad PG4s with loop length 3. There was a strong overlap of 27% ($p < 2.2e\text{-}308$) between sequences in the human and mouse genomes, however, suggesting that at least for this pattern, the PG4 populations of these genomes are comparable. The more complex the PG4 motifs become, however, the more likely it is that these subpopulations will be very different. There are clear differences in dinucleotide content between different genomes, which are often a result of differences in amino acid composition of proteins, or other environmental factors such as temperature. For genomes whose last common ancestor with *Homo sapiens* was longer ago, this divergence may be much greater. These systemic differences between may result in patterns to which the model has not previously been exposed, and reduce the performance of the model. G4Seeqer, or any other models which are trained on sequences from a single genome, should therefore be used with caution on others.

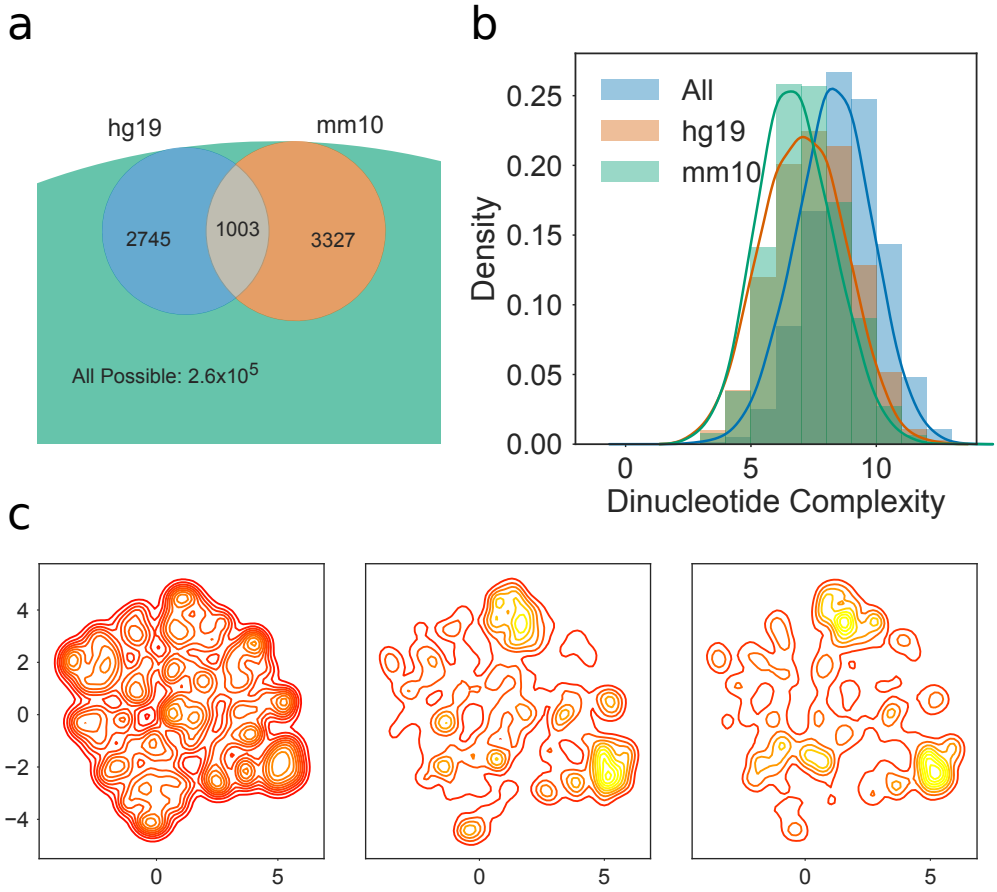

Figure 1.12: Human and mouse genomes contain different G4 subpopulations a) Venn diagram showing overlap of 3 tetrad Quadparser motifs populations with loop lengths of 3bp in the human (hg19) and mouse (mm10) genomes, compared to all possible sequences. **b)** Histogram and kernel density estimate of dinucleotide complexity for human, mouse and all possible 3 tetrad Quadparser motifs with loop length of 3bp. **c)** 2D Kernel density estimate plot showing distribution of all possible tetrad Quadparser motifs (left), those found in the human genome (centre) and those found in the mouse genome. Dimensionality reduction was conducted using UMAP with Hamming distance as the metric.

1.3.11 Conclusion

We present G4Seeqer, the first Convolutional/Recurrent Neural Network model for prediction of G Quadruplex forming structures. G4Seeqer is implemented in Python, using Cython for speed-up of G4Hunter candidate region proposal, and `Keras` with `Tensorflow` backend for neural network prediction (Abadi et al., 2016; Chollet, 2018). Weights have been trained on the G4Seq dataset (Chambers et al., 2015) and transfer learned to the rG4Seq dataset (Kwok et al., 2016), to produce models tailored for DNA and RNA G4s, respectively. It is able to process the whole human genome in approximately 1 hour on a 8 core i7 desktop computer with 16GB RAM. Because G4Seeqer is trained directly upon sequences from the human genome, rather than on derived sequence features, it is able to identify patterns in the G4Seq dataset that have not previously been reported, as well as removing false positive sequences which are flagged by pattern matching techniques. This greatly improves the accuracy of the model on various *in vitro* and *in vivo* datasets, from stabilities determined by UV melting to genomic regions identified by BG4 ChIP-seq (Hänsel-Hertsch et al., 2016).

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

Aho, A.V., Kernighan, B.W., and Weinberger, P.J. (1988). The AWK programming language (Addison-Wesley Pub. Co).

Alipanahi, B., Delong, A., Weirauch, M.T., and Frey, B.J. (2015). Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nature Biotechnology *33*, 831–838.

Andrew Collette (2013). Python and HDF5 (O'Reilly Media).

Bedrat, A., Lacroix, L., and Mergny, J.-L. (2016). Re-evaluation of G-quadruplex propensity with G4Hunter. Nucleic Acids Research *44*, 1746–1759.

Chambers, V.S., Marsico, G., Boutell, J.M., Di Antonio, M., Smith, G.P., and Balasubramanian, S. (2015). High-throughput sequencing of DNA G-quadruplex structures in the human genome. Nature Biotechnology *33*, 877–881.

Chollet, F. (2018). Keras.

Collie, G.W., Haider, S.M., Neidle, S., and Parkinson, G.N. (2010). A crystallographic and modelling study of a human telomeric RNA (TERRA) quadruplex. Nucleic Acids Research *38*, 5569–5580.

Eddy, J., and Maizels, N. (2006). Gene function correlates with potential for G4 DNA formation in the human genome. Nucleic Acids Research *34*, 3887–3896.

Ernst, J., and Kellis, M. (2017). Chromatin-state discovery and genome annotation with ChromHMM. Nature Protocols *12*, 2478–2492.

Garant, J.M., Perreault, J.P., and Scott, M.S. (2017). Motif independent identification of potential RNA G-quadruplexes by G4RNA screener. Bioinformatics *33*, 3532–3537.

Guédin, A., Gros, J., Alberti, P., and Mergny, J.-L. (2010). How long is too long? Effects of loop size on G-quadruplex stability. Nucleic Acids Research

38, 7858–7868.

Harkness, R.W., and Mittermaier, A.K. (2016). G-register exchange dynamics in guanine quadruplexes. Nucleic Acids Research *44*, 3481–3494.

Hänsel-Hertsch, R., Beraldi, D., Lensing, S.V., Marsico, G., Zyner, K., Parry, A., Di Antonio, M., Pike, J., Kimura, H., Narita, M., et al. (2016). G-quadruplex structures mark human regulatory chromatin. Nature Genetics *48*, 1267–1272.

Hon, J., Martínek, T., Zendulka, J., and Lexa, M. (2017). pqsfinder: an exhaustive and imperfection-tolerant search tool for potential quadruplex-forming sequences in R. Bioinformatics *33*, 3373–3379.

Hou, X.-M., Fu, Y.-B., Wu, W.-Q., Wang, L., Teng, F.-Y., Xie, P., Wang, P.-Y., and Xi, X.-G. (2017). Involvement of G-triplex and G-hairpin in the multi-pathway folding of human telomeric G-quadruplex. Nucleic Acids Research *45*, 11401–11412.

Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. Computing in Science and Engineering *9*, 99–104.

Huppert, J.L., and Balasubramanian, S. (2005). Prevalence of quadruplexes in the human genome. Nucleic Acids Research *33*, 2908–2916.

Jones, E., Oliphant, T., and Peterson, P. (2001). SciPy: Open source scientific tools for Python.

Kwok, C.K., Marsico, G., Sahakyan, A.B., Chambers, V.S., and Balasubramanian, S. (2016). rG4-seq reveals widespread formation of G-quadruplex structures in the human transcriptome. Nature Methods *13*, 841–844.

Lemaître, G., Nogueira, F., and Aridas, C.K. (2001). Journal of machine learning research : JMLR. The Journal of Machine Learning Research *18*, 559–563.

McInnes, L., and Healy, J. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv.

Mckinney, W. (2011). pandas: a Foundational Python Library for Data Analysis and Statistics. Python for High Performance and Scientific Computing.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research *12*, 2825–2830.

Quang, D., and Xie, X. (2016). DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. Nucleic Acids Research *44*, e107–e107.

Quinlan, A.R., and Hall, I.M. (2010). BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics (Oxford, England) *26*, 841–842.

Sahakyan, A.B., Chambers, V.S., Marsico, G., Santner, T., Di Antonio, M., and Balasubramanian, S. (2017). Machine learning model for sequence-driven DNA G-quadruplex formation. Scientific Reports *7*, 14535.

VanRossum, G. (1995). Python tutorial. Department of Computer Science [CS].

Waskom, M., Botvinnik, O., Hobson, P., Cole, J.B., Halchenko, Y., Hoyer, S., Miles, A., Augspurger, T., Yarkoni, T., Megies, T., et al. (2014). seaborn: v0.5.0 (November 2014).

Zerbino, D.R., Wilder, S.P., Johnson, N., Juettemann, T., and Flicek, P.R. (2015). The ensembl regulatory build. Genome Biology *16*, 56.