THE UNIVERSITY of EDINBURGH

MLPScore: A Neural Network-based Scoring Function for Classification of Receptor-Ligand Complexes

Biological Sciences BSc (Biochemistry Hons) 2021

Supervisor: Dr. Douglas Houston

ABSTRACT WORD COUNT:189/200; REPORT WORD COUNT: 5000/5000

Abstract:

Techniques used in drug discovery, such as high throughput screening, are highly costly. Computational approaches, such as predicting receptor-ligand interactions with scoring functions, are a potential solution for these high costs. Increasing the predictive accuracy of scoring functions is essential for improving the drug discovery process. Recently, machine learning methods have been shown to increase scoring function performance. Presented here, is a high-performance multilayer neural network-based scoring function (MLPScore) for classification of receptor-ligand complexes as strong or weak binders. In an attempt to avoid hidden biases in decoy datasets, we present a novel approach to scoring function training, using property matched decoy ligands generated using DeepCoy. Use of DeepCoy allowed MLPScore to be trained on a significantly larger dataset than previous scoring functions. MLPScore achieved an AUCPR of 0.90 and an AUROC of 0.97, demonstrating high precision and an impressive ability to distinguish between strong and weak binders. While MLPScore shows high levels of performance, our results indicate decoy bias may still remain an issue in scoring function training. This study provides an improvement on previous scoring functions and insight into how to further improve prediction of receptor-ligand interactions.

Abbreviations

SBVS – Structure-based virtual screening; SF – Scoring function; ML – Machine Learning; PCA – Principal Component Analysis; Standard Error of Mean – SEM; AUCRP – Area Under the Curve of Precision-Recall; ROC – Receiver Operating Characteristic; AUROC – Area Under the Receiver Operating Characteristic Curve

Table of Contents

1 Introduction

In an attempt to reduce the extensive financial, time and labour costs of performing high throughput screening, computational approaches have become an essential part of the lead identification process in drug discovery[1]. Improving the quality of *in silico* approaches, such as structure-based virtual screening (SBVS), is vital for increasing efficiency in the drug discovery process. In SBVS, compound libraries are screened against a target receptor for prospective binders. Ligands are docked into an allocated binding pocket on the target receptor, and a scoring function (SF) is used to predict the strength of the binding interaction. In this study, we produce a SF for predicting binary classification of receptor-ligand interactions, MLPScore, using a neural network-based approach. The purpose of this SF is to identify high affinity drug-like ligands for target receptors in SBVS.

1.1 Machine Learning Scoring Functions

Over the last decade, various machine learning (ML) approaches have been applied to SFs and have shown to be an improvement over classical physics-based linear regression predecessors[2-4]. This can be seen in Table 1 below, with ML-methods demonstrating greater Pearson's Correlation Coefficients (ρ) between predicted and actual binding affinities than their classical counterparts.

Table 1: Scoring Function Performances on the CASF-2007 Test Set

Machine learning (marked with "x") and classical scoring functions, sorted by ρ performance on the CASF-F 2007 Test Set. Adapted from Li, H. J., et al.[4]

Scoring Function	Model	Test Set ρ
AGL-Score (x)	Gradient Boosted Decision Trees	0.830
TNet-BP (x)	Convolutional Neural Networks	0.826
RF-Score v2 (x)	Random Forest	0.803
CScore (x)	Neural Networks	0.801
X-Score	Classical Linear	0.644
GOLD:ASP	Classical Linear	0.534
GlideScore-XP	Classical Linear	0.457

Random Forest[5], Support Vector Machines[6] and Convolutional Neural Networks[7] are a few examples of different ML methods that have been used to develop SFs. Neural networks in particular have been shown to be a highly promising approach in computational drug discovery, including in predicting receptor-ligand interactions[8]. NNScore[9], a neural network-based classification SF with a single hidden layer, outperformed classical SFs at its inception. More recently, Hassan et al.[10] used deep learning neural networks to produce a regression model for predicting receptor-ligand *pKd*, DLScore, which showed significantly improved performance over previous ML-based SFs. Given the demonstrated potential of neural networks in drug discovery, we decided to implement a deep learning approach to receptor-ligand classification.

1.2 Regression and Classification

ML-based SFs, such as those shown in Table 1, commonly employ a regression-based approach, predicting the *pKd* value of a receptor-ligand interaction. In a binary classification model, the receptor-ligand interaction is predicted to be either “stronger” or “weaker” than a specified threshold *pKd*. It has been suggested that a classification approach produces fewer false positive results[11]. Additionally, to our knowledge, fewer ML classification SFs have been attempted despite still being potentially useful predictors in lead identification[12]. As such, we focused on a classification-based approach.

1.3 Neural Networks

The design of neural networks is based on the arrangement of neuronal cells[13]. A representation of a neural network is shown below in Figure 1. Nodes receive and process signals before passing the resultant output on. Nodes are joined to each other by weighted connections, which allow signals to pass from one node to another. The weight of a connection determines the strength of the signal. Nodes are placed into

layers, with the nodes in one layer connected to the nodes in the adjacent layers. There are three types of layer structure: the input layer is the variable data given to the network, the output layer is the resultant prediction and hidden layers lie between the two, receiving data from one layer before processing and passing it on to the next layer.

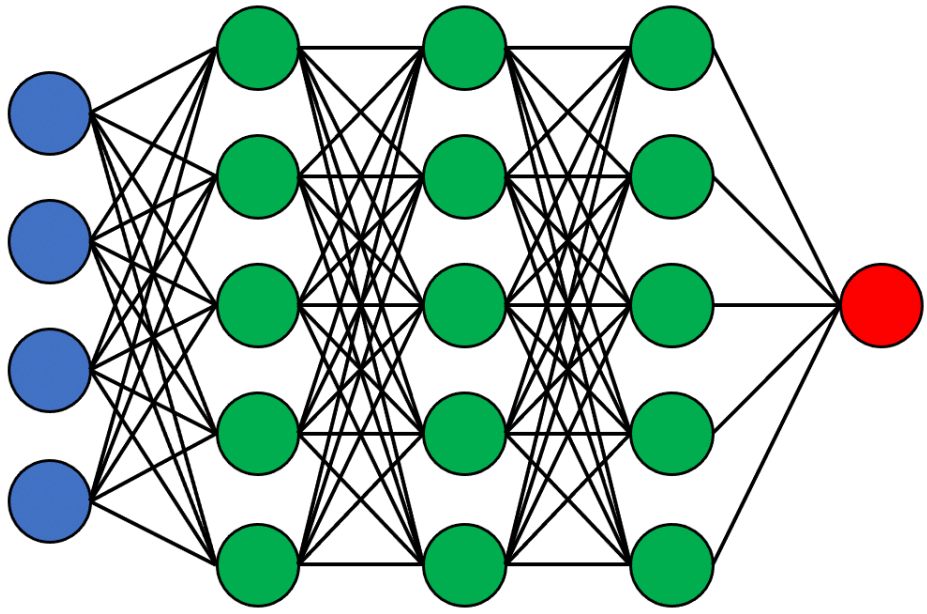


Figure 1: Diagrammatic Representation of a Neural Network
Nodes and connections are depicted as circles and lines, respectively. The input layer is shown in blue, hidden layers in green and the output layer in red.

In a feedforward neural network (also called a multilayer perceptron) the signal can only propagate in one direction. In a fully connected neural network, all the nodes in one layer are connected to all the nodes in the next layer. A neural network is referred to as ‘deep’ if it possesses more than one hidden layer.

For learning, networks are repeatedly shown data with a known outcome (the training set) and connection weights are iteratively adjusted to improve prediction accuracy. To assess how changing network parameters, such as the number layers, affects performance, a subset of the training data is used to validate performance (the validation set). This allows network parameters to be optimised for training. The

optimal model is then trained, and performance is evaluated on unseen data (the test set).

1.4 Training and Testing Data

For training and testing a ML SF for predicting binding affinities, data on receptor-ligand interactions and their respective *Kd* values are required. Interaction data can be acquired from receptor-ligand structures using software packages such as BINANA[14] and ECIF[15]. BINANA generates tallies of receptor-ligand atom contacts within 2.5 Å and 4.0 Å, summed electrostatics, hydrogen bonds and the number of ligand rotatable bonds in addition to other descriptors. ECIF generates tallies of receptor-ligand atom contacts within 6Å which includes further details, such as atom valence, attached heavy atoms, attached hydrogens and aromaticity.

Databases of active receptor-ligand crystal structures and their associated binding data, like PDBbind[16] and Iridium[17], are useful resources for the training and testing sets that ML SFs require. However, these databases often lack sufficient examples of weak binders for training. There are examples of databases with experimentally verified inactive receptor-ligand pairs, however they remain limited in their data availability. For example, LIT-PCBA[18] contains confirmed inactive ligands for only 15 target receptors. To deal with this, researchers often use docked “decoy” ligands as examples of weak binders from databases such as DUD-E[19] and DEKOIS[20]. Decoys in these databases are designed to mimic the physiochemical properties of their active counterparts but differ in terms of 2D topology, preventing any binding interaction. However, it has been demonstrated decoys from these databases possess underlying structural biases[21]. Use of these decoys in training causes ML SFs to learn the inherent biases in the datasets rather than the underlying patterns of intermolecular binding, which negatively affects SF performance in virtual screening.

A recently developed deep learning model, DeepCoy[22], claims to provide a method for generating property matched decoys from active ligands, without introducing the inherent structural biases present in decoy datasets. The use of a model like DeepCoy in generating a training set for a ML SF is a novel approach which sets our research apart from other SFs. DeepCoy also has the capacity to produce a large number of decoys in a very short timeframe, which presents an opportunity to train a ML SF on a significantly larger dataset than previous SFs.

Herein, we present a deep fully connected multilayer perceptron SF trained on a dataset of over 20,000 ligands, for high performance classification of protein-ligand binding affinities.

2 Materials and Methods

2.1 Workflow

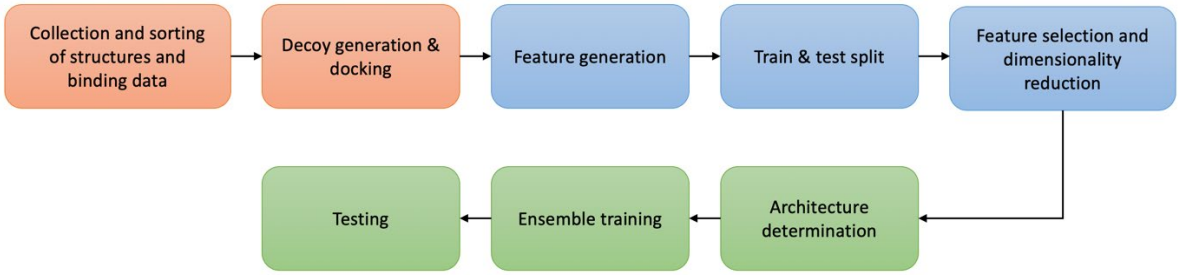


Figure 2: MLPScore Methodology Workflow
Schematic of the steps involved in developing MLPScore

2.2 Collection of Receptor-Ligand Crystal Structures and Affinities

Crystal structures of receptor-ligand complexes and their respective experimentally determined binding affinities were acquired from PDBbind[16] (4,854 structures) and Iridium[17] (120 structures). Version 2019 of the refined set from PDBbind and the Highly Trustworthy dataset from Iridium were used to ensure high quality. Wherever there was overlap between the datasets, structures from Iridium were preferentially kept. To

ensure included ligands were small drug-like molecules: 1) ligands with more than 13 rotatable bonds were identified using MGLTools 1.5.6[23] 2) ligands with a molecular weight of over 500 Da were identified using OpenBabel 3.2[24] and ODDT 0.7[25] 3) peptide containing ligands were identified using a custom python script with BioPython 1.78[26]. Any receptor-ligand structures which met these criteria were discarded. Structures with ligands incompatible with DeepCoy[22] and structures for which GWOVina 1.0[27] was unable to successfully dock decoys were also discarded. As we were only interested in binding pockets, amino acids more than 14 Å from the ligand in each structure were removed using BioPython. Ligands and receptors were converted into pdbqt file format using MGLTools scripts, “prepare_ligand4.py” and “prepare_receptor4.py”. During conversion, polar hydrogens and Gasteiger charges were added to both, and waters were removed from receptors.

Binding data for the structures were either in Kd, Ka or IC50 format, with varying unit SI prefixes. Kd and Ka values were all converted into μM Kd. Micromolar Kd was approximated from IC50 values using the Cheng-Prusoff equation[28]: $Kd = \frac{IC50}{1+\frac{[L]}{Km}}$. As we were performing binary classification rather than regression, the exact Kd did not need to be identified. As the denominator of the Cheng-Prusoff equation is always ≥ 1, for all structures with IC50 values ≤ 25 μM it could be assumed that Kd is also ≤ 25 μM. For the same reason, IC50 values > 25 μM are not guaranteed to have Kd > 25 μM. To maximise the amount training data while reducing the probability of false negative labels, we excluded structures with IC50 within the range of 25 μM < x ≤ 250 μM. All structures with a Kd ≤ 25 μM were assigned a label of 1, indicating a strong binder, and all structures with a Kd < 25 μM were assigned a label of 0, indicating a weak binder.

2.3 Property Matched Decoy Generation

RDKit 2021.03[29] was used to create SMILES format copies from active ligand pdb files for decoy generation. As DeepCoy[22] has separate pretrained models for generating decoys for phosphorus containing ligands and non-phosphorus ligands, the active ligand SMILES were split into phosphorus and non-phosphorous groups. Ligand SMILES were prepared for decoy generation using the included “prepare_data.py” python script. For each active ligand, we used DeepCoy to generate 1,000 property matched decoy ligands. Decoys for non-phosphorus actives were generated using the pretrained model, “DeepCoy_DUDE_model_e09.pickle”. Decoys for phosphorus containing actives were generated using the pretrained model, “DeepCoy_DUDE_phosphorus_model_e10.pickle”. Decoy SMILES were evaluated using the included “select_and_evaluate_decoys.py” script, selecting best 5 decoys for each active. Selected decoys were converted to pqdbt format using “prepare_ligand4.py” from MGLTools[23] and docked using GWOVina[27] into the receptor binding pocket of their respective active based on the location of the active. The number of rotatable bonds and the molecular weight of the decoys was checked using MGLTools[23] and Openbabel[24]. Polar hydrogens and gasteiger charges were added to the decoys during pdbqt conversion. Decoys were assigned a 0 label for binding strength.

The final dataset contained 3,670 active receptor-ligand pairs with 5 docked decoys for each active, totalling 22,020 ligands. Out of the active ligands, 3,587 were from PDBbind and 83 were from Iridium, 2,836 were strong binders and 834 were weak binders.

2.4 Feature Generation

To generate intermolecular interactions features for each receptor-ligand complex, we used BINANA 1.3[14] and ECIF[15].

For BINANA, pdbqt formats of receptors and ligands were used. A custom python script was used to automate running BINANA for every receptor-ligand pair. A total of 488 BINANA features were generated for each protein-ligand complex.

For ECIF, pdb format files of receptors and ligands sdf format files, created using Openbabel[24] and ODDT[25] with all hydrogens added, were used. ECIF python functions were collated into a script and run on each receptor-ligand pair. A total of 1540 ECIF features were generated for each protein-ligand complex.

BINANA, ECIF descriptors (2028 descriptors in total), as well as receptor-ligand binding data and labels were combined into a data frame in hdf5 format.

2.5 Stratified Splitting into Train and Test Sets

Actives were split using StratifiedShuffleSplit from SciKit-learn 0.22[30] into a training set with 95% of the actives and a testing set with 5% of the actives. Splitting was stratified by label, Kd and crystal resolutions from the Protein Data Bank[31] API. This was done to prevent further label imbalance in either dataset and to preserve a range of receptor-ligand structure qualities and Kd strengths for training. This was done prior to feature selection to prevent test set bias. Decoys for actives in the test set were added to the test set and decoys for actives in the training set were added to the training set. This left a training set of 20,919 receptor-ligand pairs and a test set of 1,101 receptor ligand pairs.

The test set was further split into; 1) the full test set with all actives and decoys 2) a test set of just the actives without the decoys 3) a balanced test set of actives with the same number of weak and strong binders.

2.6 Feature Selection and Dimensionality Reduction

Feature selection was performed to remove redundant features. Features with 0 variance were removed. Pearson's Correlation Coefficient was used to determine feature co-correlation and feature-label correlation. Where two features had Pearson's Coefficient > 0.9 to each other, the feature with a lower Pearson's Coefficient to the binary label was removed. This left 1,322 features. Features were then scaled using MaxAbsScaler from SciKit-learn[30] to preserve sparsity.

RFECV from SciKit-learn[30] with a Random Forest model and 5-fold cross-validation was used to perform recursive feature elimination and cross-validated selection in order to rank features and determine the optimal number of features to use (Supplementary Figure 1). The top 843 features were kept for dimensionality reduction.

ML techniques are often affected by 'the curse of dimensionality', where increasing the number of features causes performance to be reduced[32]. Dimensionality reduction is a method for counter-acting this, reducing data complexity and thereby making it more easily interpretable for the neural network[33]. Dimensionality reduction was performed using Principal Component Analysis (PCA). PCA was performed using SciKit-learn's[30] PCA method with a n_components of 0.95, giving 354 principal components to use as model inputs (Supplementary Figure 2).

2.7 Neural Network Construction

All neural networks were constructed using Keras 2.4.3[34], Tensorflow 2.4.1[35] and SciKit-learn[30]. All hidden layers used were Dense layers. Input dropout of 0.4 and hidden dropout of 0.2 were used to reduce overfitting. The remaining training parameters used were the default Keras[34] parameters for binary classification (Section 6.2). Early stopping with a patience of 30 and a batch size of 16 were used.

2.8 Architecture Determination

45 architectures with either 100, 250, 500, 750 or 1000 nodes per hidden layer, with a range of 2 to 10 hidden layers were each trained 10 times. The number of nodes between layers within each network was not varied due to time constraints. Mean and standard error of mean (SEM) for area under the curve of precision-recall (AUCPR) and area under the receive operating characteristic curve (AUROC) were calculated on a 10% training set validation split.

2.9 Ensemble Training

100 networks of the top performing architecture were trained. For each network, ModelCheckpoint from Keras[34] was used to save models with the highest validation split AUCPR. To weight networks by their performance, network predictions on the validation split were fitted to a logistic regression model using SciKit-learn[30]. The number of models used to fit the logistic regression model was varied.

2.10 Testing

Ensemble networks were used to predict scores on three different test sets (full, actives only, and balanced). Top 24 networks from NNScore[9] were used to predict scores on the three test sets. As NNScore only takes specific atom types, not all test set receptor-ligand pairs could be run on NNScore. NNScore predictions were converted by adding 1 and dividing by 2, such that the predicted score threshold for strong and weak binders was 0.5. AUCPR, and AUROC were determined in using SciKit-learn[30].

3 Results and Discussion

3.1 Architecture Variation

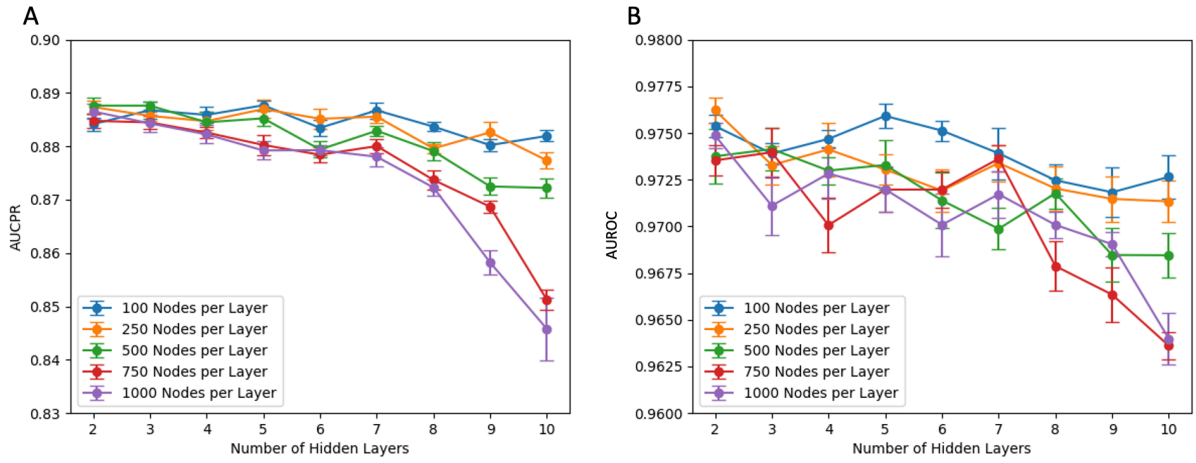


Figure 3: Architecture Mean Validation Split AUCPR and AUROC

A) Mean validation split AUCPR as a function of the number of hidden layers. B) Mean validation split AUROC as a function of the number of hidden layers. Error bars are the Standard Error of Mean and n = 10.

For this section architectures will be referred to like so: [nodes, layers]

Network architecture had a noticeable effect on performance (Figure 3). The general trend with increasing numbers of the layers and nodes was that past a certain point it led to an overall decrease in both AUCPR (Figure 3A) and AUROC (Figure 3B). The two architectures with the lowest AUCPR by a significant margin, were [750, 10] and [1000, 10]. [750, 10] and [1000, 10] showed over a 4% decrease in AUCPR when compared to the highest mean value. AUROC of [750, 10] and [1000, 10] also decreased by 1.3% from the highest mean value. No architecture showed the highest AUCPR or AUROC by a significant margin. [100, 5] had the highest mean AUCPR by a difference of 0.0004 and the second highest mean AUROC with, 99.97% of the highest mean value. [250, 2] achieved the second mean highest AUCPR, with 99.95% of the top value, and the highest mean AUROC by a difference of 0.0003. [100, 5] and [250, 2] both had overlapping SEM with 18.2% other architectures. For AUROC, [100, 5] and [250, 2] both had overlapping SEM with 13.6% of the other architectures.

The downward trend of validation set AUCPR and AUROC with increasing neural network size, with more than 250 nodes and more than 4 layers, is likely due to increasing network overfitting to the training set as architecture complexity increases. This is a common issue in neural networks with limited data availability[36]. Overlapping SEM values for top performing architectures demonstrates that there is no clear architecture which outperforms any other. For further experiments I decided to use [100, 5]. Due to overlapping SEM, multiple other architectures, such as [250, 2], would have been appropriate to choose based on the results.

3.2 Ensemble Approach

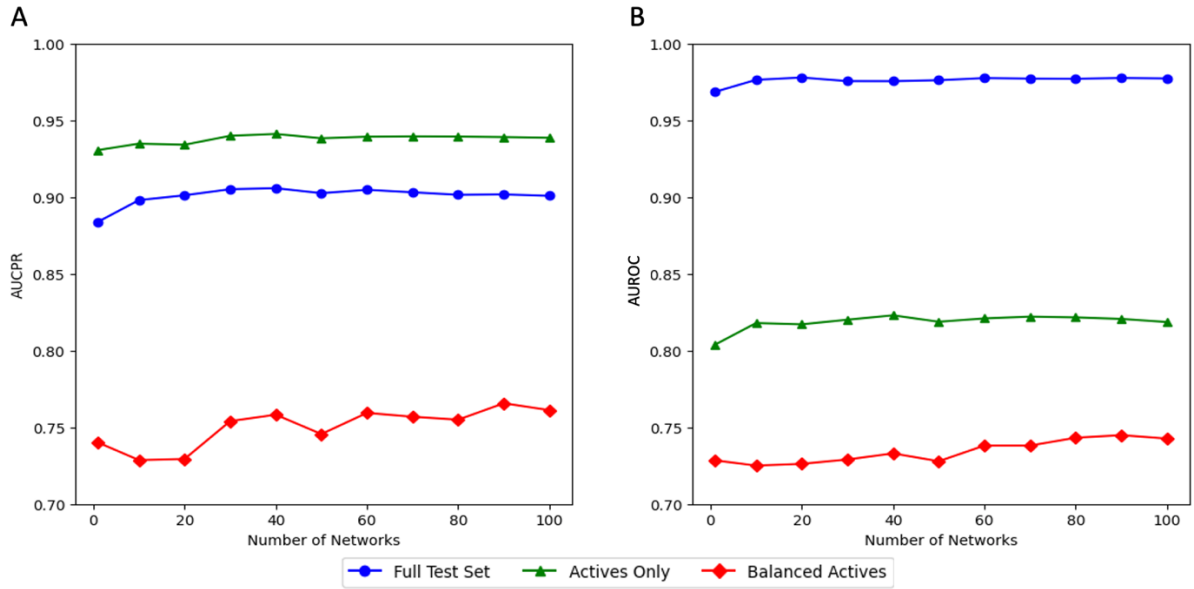


Figure 4: AUCPR and AUROC as a Function of the Number of Networks
Ensemble performance using logistic regression to weight networks. A) AUCPR of network ensembles on the three different test sets as a function of the number of networks. B) AUROC of network ensembles on the three different test sets as a function of the number of networks.

Using multiple networks in an ensemble approach increased AUCPR and AUROC in all three test sets compared to using the top network alone (Figure 4). The top performing network had an AUCPR of 0.884 and an AUROC of 0.969 on the full test set. All ensembles with 20 or more networks had an AUCPR of over 0.90 and all

ensembles with 10 or more networks had an AUROC of over 0.97. This trend was seen in the actives only test set, with AUCPR and AUROC increasing by up to 1.06% and 1.91% respectively. Balanced test set AUCPR and AUROC increased by over 1.5%.

Although changes in AUCPR and AUROC were low, the increases were clear in all three test sets. This suggests that using multiple networks does indeed improve predictive performance of MLPScore. These results are consistent with other scoring functions. The accuracy of both NNScore[9] and DLScore[10] was greater when combining the predictions of multiple networks.

3.3 MLPScore Outperforms NNScore

On the full test set, MLPScore showed higher AUCPR and AUROC scores than NNScore[9] (Figure 5). MLPScore achieved an AUCPR 2.30 times greater than NNScore, representing a 0.40 increase (Figure 5A). MPLScore maintained 100% precision for over 20% recall, whereas precision for NNScore fell under 60% within 10% recall. Similarly, AUROC increased by a factor of 1.24 from NNScore to MLPScore. MLPScore demonstrated a 95.6% increase over random guessing in AUROC, equivalent to 97.8% of the AUROC of a perfect model (Figure 5B). By comparison, NNScore achieved a 57.4% AUROC increase over random guessing, 78.7% of the AUROC of a perfect model. At very low FPR MLPScore has higher TPR, with over 90% TPR at a 5% FPR, compared to under 40% TPR at a 5% FPR for NNScore (Figure 5B).

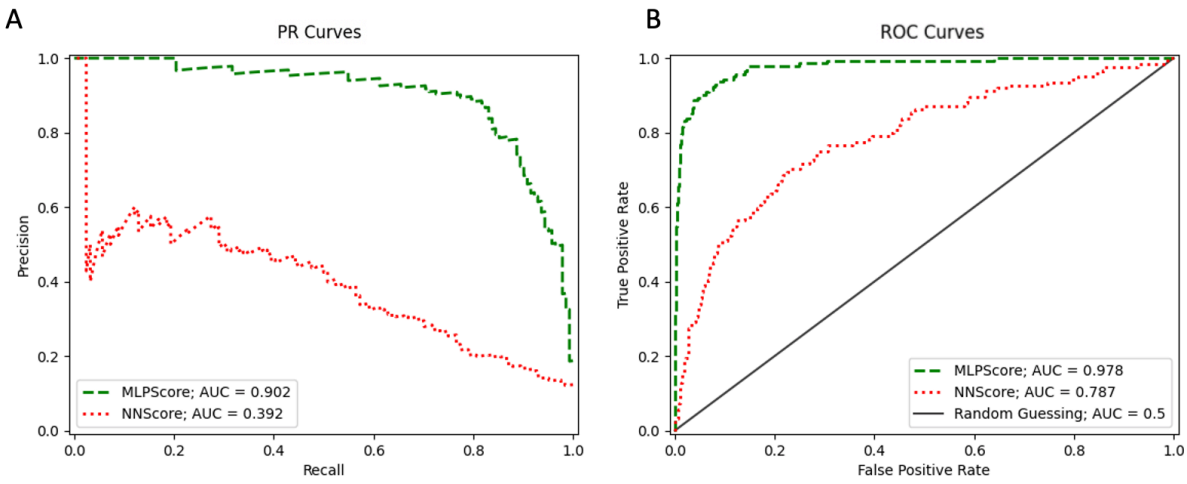


Figure 5: PR and ROC Curves of MLPScore and NNScore

A) Precision-Recall curves with AUC of top 90 MLPScore networks (green, dashed) and NNScore top 24 networks (red, dotted) on the full test set. B) Receiver Operating Characteristic curves of top 90 MLPScore networks (green, dashed) NNScore top 24 networks (red, dotted) on the full test set. The line y = x is equivalent to random guessing of the label.

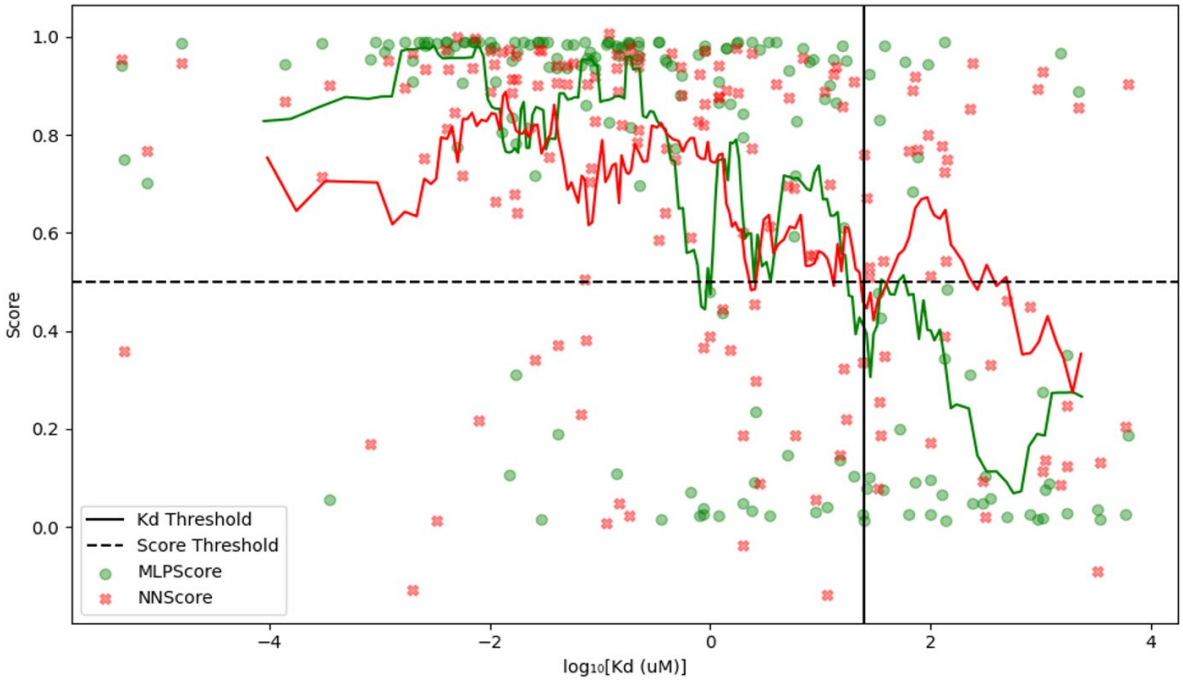


Figure 6: Scatter Plot of Predicted Scores against Binding Affinity

Scatter plot with test set active predictions on the y-axis and $\log_{10}$ of micromolar Kd on the x-axis. Scores by top 90 MLPScore networks are shown as green dots and scores by top 24 NNScore networks are shown as red crosses. Moving averages over 10 points are shown as line plots for MLPScore (green) and NNScore (red). The dashed line y = 0.5 is the score prediction threshold for strong and weak binders. Below this line are weak predictions, and above this line are strong predictions. The solid black vertical line, x = $\log_{10}(25)$, is the Kd threshold for true strong and weak binders. To the right of this line are true weak binders, and to the left of this line are true strong binders.

When predicting the active test set, MLPScore showed a decrease in the rate of false positives and false negatives compared to NNScore (Figure 6). Only 7.9% of positive predictions by MLPScore were false, while the false positive rate of NNScore was 2.2 times greater, with 17.4% of positive predictions being false. MLPScore had a false negative rate of 43.1% compared to 58.3% by NNScore. These trends can be seen in the difference between the moving averages shown in Figure 6. When a regression is fit to the moving averages using polyfit from NumPy[37], MLPScore has a gradient with double the magnitude of NNScores gradient, -0.125 compared to -0.0611.

These data collectively show that MLPScore outperforms NNScore in every aspect. The greater AUCPR of MLPScore suggests MLPScore is both highly precise and maintains a high level of recall in comparison to NNScore. The higher levels of precision of MLPScore are confirmed by the lower false positive rate of MLPScore when predicting on the actives only test set, which similarly suggests that MLPScore is a considerably more precise SF than NNScore. Additionally, the greater levels of recall of MLPScore are further demonstrated by the lower false negative rates on the active test set. The difference in AUROC on the full test set indicates that MLPScore is able to differentiate between strong and weak binders with far greater accuracy than NNScore and is therefore a much better classifier[38]. In fact, the AUROC scores on the full test set suggest that MLPScore is a near-perfect model, achieving almost 100% AUROC. The greater magnitude of gradient for MLPScore in Figure 6, further suggests that MLPScore has a superior ability to distinguish between classes when compared to NNScore.

3.4 Decoy Bias

MLPScore showed decreased levels of performance compared to the full test set when tested on the actives only and balanced test sets (Figure 7A). On the actives only and

balanced test sets, MLPScore AUROC decreased from the full test set by 16.1% and 23.8% respectively. However, AUROC remained greater than random guessing by a factor of 1.64 and 1.49 in each case. A reduction was also seen for NNScore from the full test set to the actives only and balanced test sets, but by a lesser proportion (Figure 7B). NNScore AUROC was 9.1% lower on the actives only test set and 14.9% lower on the balanced active test set. Similarly to MLPScore, NNScore still had a greater AUROC than random guessing, with AUROC 43.6% greater on the actives only test set and 34.0% greater on the balanced test set. MLPScore had a greater AUROC score than NNScore on all test sets by factors of 1.24, 1.15 and 1.11 respectively.

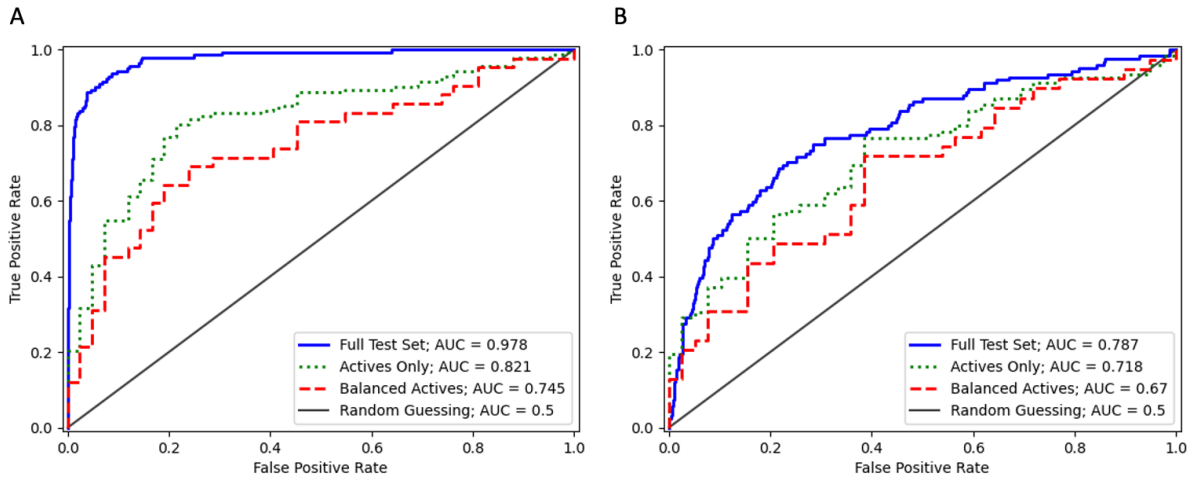


Figure 7: MLPScore and NNScore ROC Curves on All Test Sets

Receiver Operating Characteristic Curves for the full test set (blue, solid), only test set actives (green, dotted) and on the balance set of actives (red, dashed). The line y = x is equivalent to random guessing of the label. Area under the curve is shown in the figure legends. A) MLPScore ROC curves. B) NNScore ROC curves.

To our knowledge, we are the first to develop a ML SF using decoys generated as described in Section 2.3. As previously mentioned in Section 1.4, use of decoy ligands from databases such as DUD-E[19] in ML training are a major cause of performance bias in SF assessment[21]. The use of DeepCoy[22] in generating property matched decoys was an attempt to avoid the issues of bias present in decoy databases.

The decreases in MLPScore performance seen when removing decoys from the test set suggest that decoy bias may still be present. The reduction in AUROC from the full test set to the actives only and balanced test sets indicate that MLPScore is worse at differentiating between strong and weak actives than between strong actives and decoys. This is possibly due to weak binders still being binders rather than inactives, making them more difficult for MLPScore to classify correctly than decoys. Lower performance may also be due to the decrease in testing set size causing outliers to appear more prominently. Nevertheless, given the decrease is so pronounced, the possibility of decoy bias being a contributing factor is extremely likely. This is suggested by the fact that although NNScore decreases in AUROC, the decrease is less notable. As NNScore was not trained on our decoys, the decrease can be attributed to factors other than decoy training bias, such as weak binders being harder to differentiate than inactives. Therefore, the greater decrease for MLPScore compared to NNScore may be due to decoy bias. It is unclear whether bias arises from underlying biases in decoy structures or from the fact that decoys are docked and active ligands are not. Despite this possible bias MLPScore still outperforms NNScore on all three training sets by a notable amount.

3.5 Looking Ahead

Using increased dataset sizes, novel decoy development and modern neural network python libraries, I was able to produce a high-performance SF classifier, which outperforms NNScore in terms of precision, recall and class differentiation. The current performance of MLPScore still needs to be validated in virtual screening, and methods for further improving performance still need to be explored.

Given the input variables used, it was found that multiple different architectures would have be appropriate to implement. However, a more detailed investigation into node

combinations may be required to verify how significant architecture is to network performance. MLPScore training would have to be repeated with more architectures, such as varying the combinations of nodes in each network (e.g. 250, 100, 100 in a three hidden layer network). This would determine whether architecture could improve performance further or if the performance limit has been reached. Additionally, use of hyperparameter tuning with Bayesian optimisation[39], random search[40], or cross-validated grid search[30,41] should be applied to MLPScore retraining for potentially significant increases in performance.

It was confirmed that an ensemble approach is beneficial to AUCRP and AUROC performance. However, as only trained 100 networks were trained this should be investigated further with more networks. Ideally at least 1000 networks should have been trained, however time constraints prevented this. Other common methodologies for improving ensemble network performance includes bagging, boosting or bootstrapping approaches to dataset preparation[42,43], which focus on maximising ensemble performance rather than individual network performance. These methodologies increase training data variation between networks, possibly improving the outcome when networks are combined. Further retraining of MLPScore using these methods to verify their validity with would be required.

A virtual screen using both MLPScore and NNScore against a target receptor that is not present in either the test set or the training set, with docked experimentally verified inactives and actives with Kd < 50 nM, similarly to previously described[3,44], should be performed in order to verify the results shown here. For example, Isocitrate dehydrogenase from LIT-PCBA[18] is a potential target for virtual screen verification. It is potential therapeutic target for brain tumours[45] with 39 known active inhibitors and 362,049 experimentally verified inactives. MLPScore performance should be further

compared by enrichment factor[46] in virtual screening to other more recently developed SFs, such as DLScore[10], BCL-DockANNScore[47] and RFScore-v3[48], to assess whether MLPScore contributes an improvement to the current state of receptor-ligand SFs. The virtual screen could be used to investigate whether the decrease in performance following removal of decoys from the test set is also apparent in screening.

To determine whether bias arises from underlying biases in decoy structures or from docking, MLPScore should be re-trained using active ligands docked in the same way as decoys. This could remove the possibility of docking introducing training biases. Additionally, increasing the number of the training data samples by using another dataset in addition to PDBbind and Iridium, such as BindingMOAD[49], could reduce the prominence of dataset biases. This could also counteract potential overfitting observed in more complex architectures. Finally, a consensus approach with another SF which uses a different ML-based algorithm, such as Random Forest[43], could serve as a method for cross-validating predictions.

To ensure that the extensive costs of lead identification are reduced, it is imperative that computational approaches to drug discovery improve. By developing a new high -performance SF for classification of receptor-ligand complexes, we believe our research is an instrumental contribution to the development for virtual screening methodologies.

4 Acknowledgements

I would like to sincerely thank everyone who offered me their support during the last few months. Above all, I would like to thank my supervisor, Doug Houston, for his time, expertise and guidance throughout the course of my Senior Honours Project. I would also like to thank Dr Vincent Blay for his advice on all aspects of the project as well as his help with decoy docking, without which this project would not have been possible. Additionally, I also owe special thanks to my friend and fellow student, Miles McGibbon, for his collaboration and support, for which I am extremely grateful. Family and friends who have provided encouragement and support during this year deserve to be mentioned as well; I appreciate everything, and I could not have achieved anywhere near as much without you.

5 References

1 Tang, Y. and Marshall, G., 2011. Virtual Screening for Lead Discovery. *Methods in Molecular Biology*, pp.1-22.

2 Shen, C., Hu, Y., Wang, Z., Zhang, X., Zhong, H., Wang, G., Yao, X., Xu, L., Cao, D. and Hou, T., 2020. Can machine learning consistently improve the scoring power of classical scoring functions? Insights into the role of machine learning in scoring functions. *Briefings in Bioinformatics*, 22(1), pp.497-514..

3 Wójcikowski, M., Ballester, P. and Siedlecki, P., 2017. Performance of machine-learning scoring functions in structure-based virtual screening. *Scientific Reports*, 7(1).

4 Li, H., Sze, K., Lu, G. and Ballester, P., 2020. Machine-learning scoring functions for structure-based drug lead optimization. *WIREs Computational Molecular Science*, 10(5).

5 Ballester, P. and Mitchell, J., 2010. A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking. *Bioinformatics*, 26(9), pp.1169-1175.

6 Kinnings, S., Liu, N., Tonge, P., Jackson, R., Xie, L. and Bourne, P., 2011. A Machine Learning-Based Method To Improve Docking Scoring Functions and Its Application to Drug Repurposing. *Journal of Chemical Information and Modeling*, 51(2), pp.408-419.

7 Cang, Z. and Wei, G., 2017. TopologyNet: Topology based deep convolutional and multi-task neural networks for biomolecular property predictions. *PLOS Computational Biology*, 13(7).

8 Chen, H., Engkvist, O., Wang, Y., Olivecrona, M. and Blaschke, T., 2018. The rise of deep learning in drug discovery. *Drug Discovery Today*, 23(6), pp.1241-1250.

9 Durrant, J. and McCammon, J., 2010. NNScore: A Neural-Network-Based Scoring Function for the Characterization of Protein−Ligand Complexes. *Journal of Chemical Information and Modeling*, 50(10), pp.1865-1871.

10 Hassan, M., Mogollon, D., Fuentes, O. and sirimulla, s., 2018. DLSCORE: A Deep Learning Model for Predicting Protein-Ligand Binding Affinities. *Abstracts of Papers of the American Chemical Society,* 255

11 Adeshina, Y., Deeds, E. and Karanicolas, J., 2020. Machine learning classification can reduce false positives in structure-based virtual screening. *Proceedings of the National Academy of Sciences*, 117(31), pp.18477-18488.

12 Ballester, P., 2019. Selecting machine-learning scoring functions for structure-based virtual screening. *Drug Discovery Today: Technologies*, 32-33, pp.81-87.

13 Krogh, A., 2008. What are artificial neural networks?. *Nature Biotechnology*, 26(2), pp.195-197.

14 Durrant, J. and McCammon, J., 2011. BINANA: A novel algorithm for ligand-binding characterization. *Journal of Molecular Graphics and Modelling*, 29(6), pp.888-893.

15 Sánchez-Cruz, N., Medina-Franco, J., Mestres, J. and Barril, X., 2020. Extended connectivity interaction features: improving binding affinity prediction through chemical description. *Bioinformatics*.

16 Wang, R., Fang, X., Lu, Y. and Wang, S., 2004. The PDBbind Database: Collection of Binding Affinities for Protein−Ligand Complexes with Known Three-Dimensional Structures. *Journal of Medicinal Chemistry*, 47(12), pp.2977-2980.

17 Warren, G., Do, T., Kelley, B., Nicholls, A. and Warren, S., 2012. Essential considerations for using protein–ligand structures in drug discovery. *Drug Discovery Today*, 17(23-24), pp.1270-1281.

18 Tran-Nguyen, V., Jacquemard, C. and Rognan, D., 2020. LIT-PCBA: An Unbiased Data Set for Machine Learning and Virtual Screening. *Journal of Chemical Information and Modeling*, 60(9), pp.4263-4273.

19 Mysinger, M., Carchia, M., Irwin, J. and Shoichet, B., 2012. Directory of Useful Decoys, Enhanced (DUD-E): Better Ligands and Decoys for Better Benchmarking. *Journal of Medicinal Chemistry*, 55(14), pp.6582-6594.

20 Bauer, M., Ibrahim, T., Vogel, S. and Boeckler, F., 2013. Evaluation and Optimization of Virtual Screening Workflows with DEKOIS 2.0 – A Public Library of Challenging Docking Benchmark Sets. *Journal of Chemical Information and Modeling*, 53(6), pp.1447-1462.

21 Chen, L., Cruz, A., Ramsey, S., Dickson, C., Duca, J., Hornak, V., Koes, D. and Kurtzman, T., 2019. Hidden bias in the DUD-E dataset leads to misleading performance of deep learning in structure-based virtual screening. *PLOS ONE*, 14(8), p.e0220113.

22 Imrie, F., Bradley, A. and Deane, C., 2021. Generating property-matched decoy molecules using deep learning. *Bioinformatics*.

23 Morris, G., Huey, R., Lindstrom, W., Sanner, M., Belew, R., Goodsell, D. and Olson, A., 2009. AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. *Journal of Computational Chemistry*, 30(16), pp.2785-2791.

24 O'Boyle, N., Banck, M., James, C., Morley, C., Vandermeersch, T. and Hutchison, G., 2011. Open Babel: An open chemical toolbox. *Journal of Cheminformatics*, 3(1).

25 Wójcikowski, M., Zielenkiewicz, P. and Siedlecki, P., 2015. Open Drug Discovery Toolkit (ODDT): a new open-source player in the drug discovery field. *Journal of Cheminformatics*, 7(1).

26 Cock, P., Antao, T., Chang, J., Chapman, B., Cox, C., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B. and de Hoon, M., 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11), pp.1422-1423.

27 Wong, K., Tai, H. and Siu, S., 2020. GWOVina: A grey wolf optimization approach to rigid and flexible receptor docking. *Chemical Biology & Drug Design*, 97(1), pp.97-110.

28 Craig, D., 1993. The Cheng-Prusoff relationship: something lost in the translation. *Trends in Pharmacological Sciences*, 14(3), pp.89-91.

29 Landrum, G., Tosco, P., Kelley, B., Vianello, R., Dalke, A., N, D., Kawashima, E., Cole, B., Turk, S., Swain, M., Cosgrove, D., Vaucher, A., Wójcikowski, M., Jones, G., Probst, D., godin, g., Scalfani, V., Pahl, A., Berenger, F., Sforna, G. and Jensen, J., 2021. rdkit/rdkit: 2021_03_1 (Q1 2021) Release. [online] Zenodo. Available at: <http://dx.doi.org/10.5281/ZENODO.4639764>.

30 Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, pp.2825-2830.

31 Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I. and Bourne, P., 2000. The Protein Data Bank. *Nucleic Acids Research*, 28(1), pp.235-242.

32 Bonetta, R. and Valentino, G., 2019. Machine learning techniques for protein function prediction. *Proteins: Structure, Function, and Bioinformatics*, 88(3), pp.397-413.

33 Deisenroth, M., Faisal, A. and Ong, C., 2020. *Mathematics for machine learning*. Cambridge University Press, pp.317-343.

34 Chollet, F., O'Malley, T., Ta, Z., Bileschi, S., Gibson, A., Allaire, J.J., Rahma, F., Branchaud-Charron, F., Lee, T., de Marmiesse, G., 2015. Keras. Available at <https://github.com/fchollet/keras>.

35 Abadi, M., Barham, P., Chen, J. M., Chen, Z. F., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X. Q., 2016. TensorFlow: A system for large-scale machine learning. *Proceedings of Osdi'16: 12th Usenix Symposium on Operating Systems Design and Implementation*, 265-283.

36 Bejani, M. and Ghatee, M., 2021. A systematic review on overfitting control in shallow and deep neural networks. *Artificial Intelligence Review*.

37 Harris, C., Millman, K., van der Walt, S., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M., Brett, M., Haldane, A., del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. and Oliphant, T., 2020. Array programming with NumPy. *Nature*, 585(7825), pp.357-362.

38 Bradley, A., 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7), pp.1145-1159.

39 Sjöberg, A., Önnheim, M., Gustavsson, E. and Jirstrand, M., 2019. Architecture-Aware Bayesian Optimization for Neural Network Tuning. *Lecture Notes in Computer Science*, pp.220-231.

40 Bergstra, J. and Y. Bengio (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 281-305.

41 Akl, A., El-Henawy, I., Salah, A. and Li, K., 2019. Optimizing deep neural networks hyperparameter positions and values. *Journal of Intelligent & Fuzzy Systems*, 37(5), pp.6665-6681.

42 Baskin, I., Winkler, D. and Tetko, I., 2016. A renaissance of neural networks in drug discovery. *Expert Opinion on Drug Discovery*, 11(8), pp.785-795.

43 Breiman, L., 2001. Random forests. *Machine Learning*, 45(1), pp.5-32.

44 de Ávila, M., Xavier, M., Pintro, V. and de Azevedo, W., 2017. Supervised machine learning techniques to predict binding affinity. A study for cyclin-dependent kinase 2. *Biochemical and Biophysical Research Communications*, 494(1-2), pp.305-310.

45 Liu, X. and Z. Q. Ling (2015). Role of isocitrate dehydrogenase 1/2 (IDH 1/2) gene mutations in human tumors. Histology and Histopathology 30(10) 1155-1160.

46 McGann, M., Nicholls, A. and Enyedy, I., 2015. The statistics of virtual screening and lead optimization. *Journal of Computer-Aided Molecular Design*, 29(10), pp.923-936.

47 Brown, B., Mendenhall, J., Geanes, A. and Meiler, J., 2021. General Purpose Structure-Based Drug Discovery Neural Network Score Functions with Human-Interpretable Pharmacophore Maps. *Journal of Chemical Information and Modeling*, 61(2), pp.603-620.

48 Li, H., Leung, K., Wong, M. and Ballester, P., 2015. Improving AutoDock Vina Using Random Forest: The Growing Accuracy of Binding Affinity Prediction by the Effective Exploitation of Larger Data Sets. *Molecular Informatics*, 34(2-3), pp.115-126.

49 Hu, L., Benson, M., Smith, R., Lerner, M. and Carlson, H., 2005. Binding MOAD (Mother Of All Databases). *Proteins: Structure, Function, and Bioinformatics*, 60(3), pp.333-340.

6 Appendix

6.1 Index of Software Unix Commands

Software	Unix Command and Options
MGLTools 1.5.6[23]	
Ligand pdbqt conversion	- prepare_ligand4.py \ -l ligand.pdb \ -A hydrogens \ -o ligand.pdbqt
Receptor pdbqt conversion	- prepare_receptor4.py \ -r protein.pdb \ -A hydrogens \ -o protein.pdbqt \ -U waters
GWOVina 1.0[27]	
Decoy Docking	GWOVina \ --receptor receptor.pdbqt \ --ligand decoy.pdbqt \ --center_x x_center* \ --center_y y_center* \ --center_z z_center* \ --size_x x_length** \ --size_y y_length** \ --size_z z_length** \ --exhaustiveness=32 \ --num_wolves=40 --num_modes=5 --energy_range=4
BINANA 1.3[14]	python \ binana_path \ -receptor receptor_path \ -ligand ligand_path \ > destination_path.txt

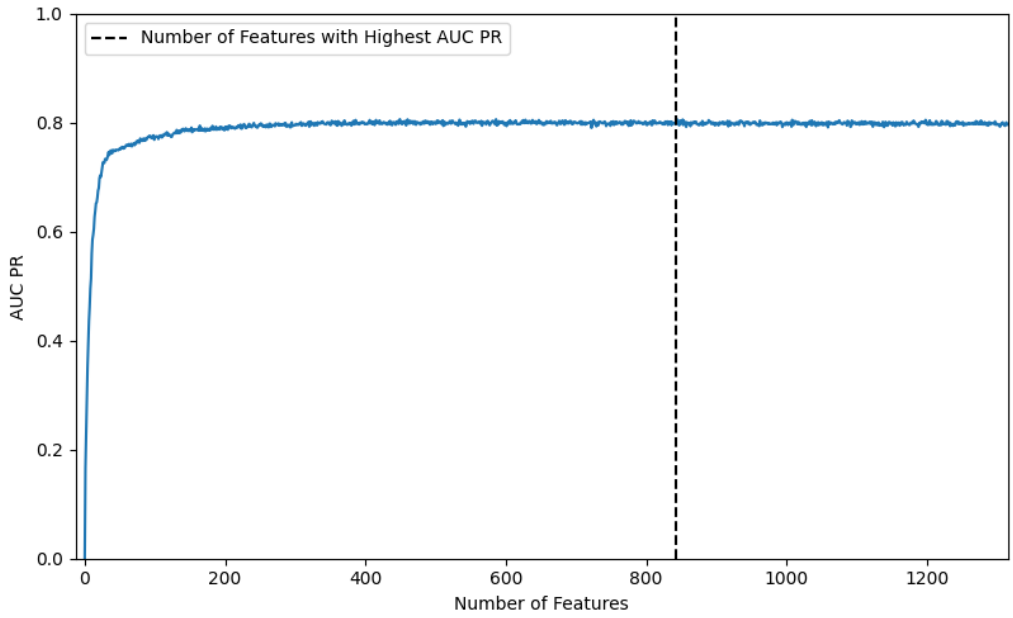
* Center coordinates = active ligand center

** Search space = Active ligand dimensions + 12.0 Å

6.2 Neural Network Hyperparameters

Parameter	Option Used
Loss Function	Binary Cross Entropy
Activation Function	ReLU
Output Layer Activation Function	Sigmoid
Optimiser	Adam
Learning Rate	0.001
Input Layer Dropout	0.4
Hidden Layer Dropout	0.2
Weight Initialiser	Glorot Uniform
Bias Initialiser	Zeros

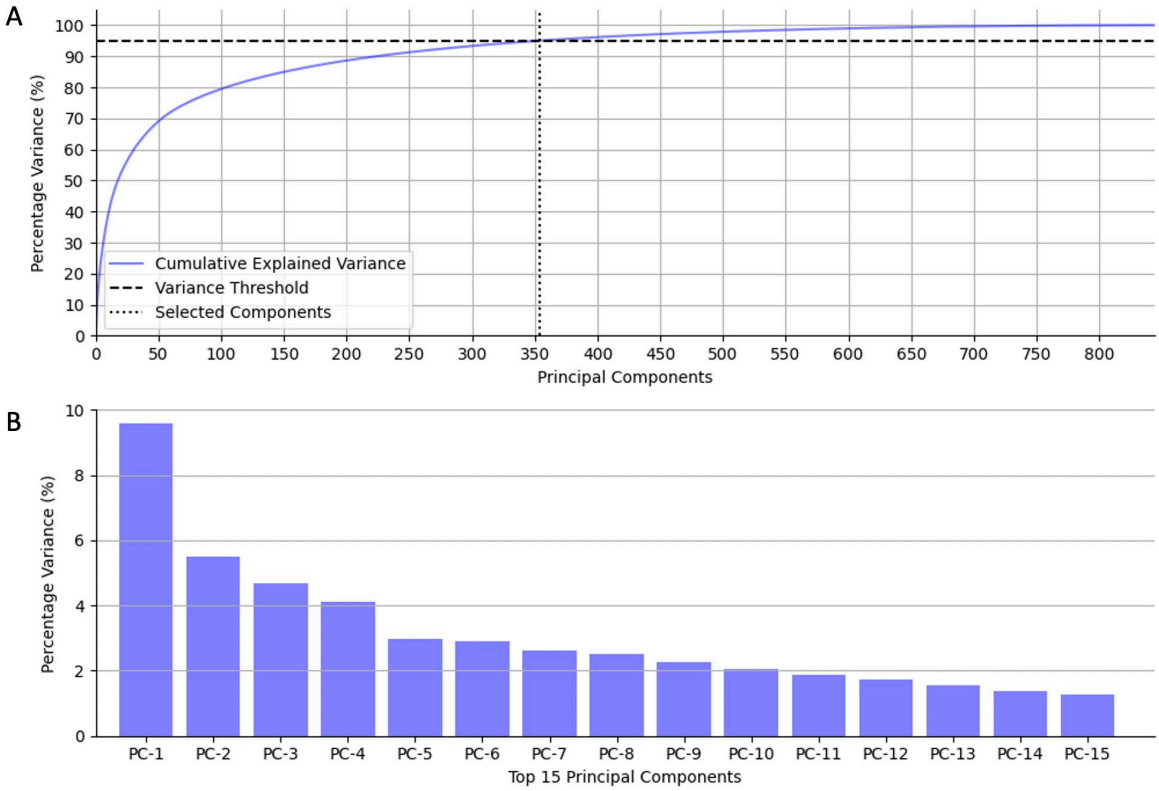
6.3 RFECV Feature Ranking



Supplementary Figure 1: RFECV AUCPR Against Number of Features

AUC PR as a function of the number of ranked features using Recursive Feature Elimination and Cross-Validated Selection with a Random Forest model and 5-fold cross validation. Dotted line denotes the number of features with the greatest AUC PR value.

6.4 Principal Component Analysis



Supplementary Figure 2: PCA Explained Variance

A) PCA cumulative percentage variance as a function of the number of components. Dashed line is the 95% threshold used for dimensionality reduction. Dotted line is the number of components selected for training (354). B) Individual explained variance as a percentage of the top 15 components.

6.5 GitHub

Link to project GitHub repository: https://github.com/smkyrle/MLPScore

6.6 PDB Codes

6.6.1 Training set:

184l, 185l, 186l, 187l, 188l, 1a28, 1a4k, 1a99, 1a9q, 1add, 1ado, 1ai4, 1ai5, 1ai7, 1aid, 1aj7, 1ajn, 1ajp, 1ajq, 1alw, 1amk, 1amw, 1atl, 1atr, 1avn, 1ax0, 1azm, 1b57, 1b8n, 1b8o, 1b8y, 1bcd, 1bcu, 1bgq, 1bju, 1bjv, 1bm7, 1bma, 1bn1, 1bn3, 1bn4, 1bnn, 1bnq, 1bnt, 1bnu, 1bnv, 1bnw, 1bp0, 1bq4, 1br6, 1bty, 1bxr, 1bzc, 1bzj, 1bzy, 1c1r, 1c1u, 1c1v, 1c3x, 1c5c, 1c5n, 1c5o, 1c5p, 1c5q, 1c5s, 1c5t, 1c5x, 1c5y, 1c83, 1c84, 1c86, 1c87, 1c88, 1cbx, 1cet, 1ciz, 1cnx, 1cps, 1ctr, 1ctt, 1ctu, 1cx2, 1d09, 1d2e, 1d4p, 1d6v, 1d7i, 1d7j, 1d9i, 1dar, 1dd7, 1det, 1df8, 1dgm, 1dhi, 1dhj, 1dl7, 1dqn, 1drj, 1drk, 1dud, 1duv, 1dy4, 1dzk, 1e1v, 1e1x, 1e2k, 1e2l, 1e3g, 1e3v, 1e4h, 1e66, 1e6q, 1e6s, 1eb2, 1ec9, 1ecq, 1ecv, 1efy, 1egh, 1ejn, 1ela, 1eld, 1ele, 1enu, 1eoc, 1erb, 1ew8, 1ew9,

1ezq, 1f0r, 1f0s, 1f0t, 1f0u, 1f3e, 1f4e, 1f4f, 1f4g, 1f4x, 1f57, 1f5k, 1f5l, 1f74, 1f8b, 1f8d, 1f8e, 1fcy, 1fcz, 1fd0, 1fh7, 1fh8, 1fh9, 1fkg, 1fkh, 1fki, 1fkw, 1fl3, 1flr, 1fm6, 1fpc, 1frp, 1ft7, 1ftm, 1fv0, 1fvt, 1fzq, 1g1d, 1g2o, 1g32, 1g36, 1g3d, 1g3e, 1g46, 1g48, 1g4o, 1g52, 1g54, 1g85, 1g98, 1gaf, 1gfy, 1ghv, 1ghw, 1gi1, 1gi4, 1gi7, 1gj6, 1gjc, 1gpk, 1gpn, 1grp, 1gwv, 1gx8, 1gyy, 1h0a, 1h1p, 1h1s, 1h22, 1h2k, 1h46, 1h4w, 1hdq, 1hdy, 1hee, 1hgh, 1hgi, 1hgj, 1hi3, 1hi4, 1hi5, 1hlk, 1hp0, 1hp5, 1hpo, 1hwi, 1hwr, 1hww, 1hyo, 1i2s, 1i37, 1i7z, 1i9n, 1i9p, 1ie9, 1if7, 1if8, 1igb, 1igj, 1iih, 1ik4, 1ikt, 1ivb, 1ivd, 1ive, 1ivf, 1iy7, 1j01, 1j14, 1j16, 1j17, 1j36, 1jak, 1jao, 1jaq, 1jd0, 1jgl, 1jla, 1jmf, 1jmg, 1jsv, 1jvu, 1jys, 1k1i, 1k1j, 1k1l, 1k1m, 1k1o, 1k21, 1k22, 1k27, 1k4g, 1k4h, 1k9s, 1kav, 1kc7, 1kdk, 1ke5, 1kel, 1km3, 1kmy, 1koj, 1kpm, 1ksn, 1kug, 1kui, 1kuk, 1kv1, 1kv5, 1kyv, 1l2s, 1l7f, 1l83, 1l8g, 1lah, 1lbf, 1lbk, 1lcp, 1lgt, 1lhu, 1li2, 1li3, 1li6, 1lke, 1lnm, 1lpz, 1lqd, 1lrh, 1lvu, 1lyx, 1m0n, 1m0o, 1m0q, 1m1b, 1m2p, 1m2q, 1m2r, 1m2x, 1m2z, 1m48, 1m5w, 1m7d, 1m7y, 1mai, 1mbi, 1mfi, 1ml1, 1mmq, 1mmr, 1mmv, 1moq, 1mrs, 1mrw, 1mrx, 1mts, 1mu6, 1mu8, 1mue, 1my4, 1mzc, 1n0s, 1n1m, 1n2j, 1n2v, 1n3i, 1n46, 1n4h, 1n4k, 1n51, 1n5r, 1nc1, 1nc3, 1ndv, 1ndw, 1ndz, 1nf8, 1nfu, 1nfw, 1nfx, 1nfy, 1nhu, 1nhz, 1nja, 1njc, 1njd, 1nje, 1nki, 1nli, 1nm6, 1no6, 1np0, 1nq7, 1nt1, 1nvq, 1nvr, 1nvs, 1nw4, 1nw5, 1nw7, 1nwl, 1o0f, 1o0h, 1o0m, 1o0n, 1o2h, 1o2j, 1o2n, 1o2o, 1o2q, 1o2r, 1o2w, 1o2z, 1o30, 1o33, 1o35, 1o36, 1o38, 1o3d, 1o3f, 1o3i, 1o3j, 1o3l, 1o5a, 1o5c, 1o5e, 1o5g, 1o5r, 1o7o, 1o86, 1oar, 1oba, 1ocq, 1od8, 1odi, 1odj, 1oe8, 1of1, 1ogd, 1ogx, 1ogz, 1oif, 1okl, 1ols, 1olu, 1olx, 1om1, 1onz, 1ork, 1os0, 1os5, 1owe, 1owh, 1oxr, 1p1n, 1p1o, 1p1q, 1p2y, 1p57, 1pa9, 1pb8, 1pb9, 1pbq, 1pdz, 1pfu, 1pgp, 1phw, 1pkx, 1pme, 1pmn, 1pot, 1ppc, 1pph, 1pr5, 1ps3, 1pvn, 1px4, 1pxn, 1pxp, 1pzi, 1pzp, 1q1g, 1q41, 1q54, 1q5k, 1q65, 1q72, 1q7a, 1q8t, 1q8u, 1q8w, 1q91, 1qan, 1qb6, 1qb9, 1qbn, 1qbo, 1qbq, 1qbv, 1qf0, 1qf1, 1qf2, 1qft, 1qhi, 1qk3, 1qk4, 1qkt, 1ql7, 1ql9, 1qy1, 1qy2, 1qyg, 1r0p, 1r1h, 1r1j, 1r58, 1r5y, 1r9l, 1rbp, 1rd4, 1rjk, 1rmz, 1rnm, 1rnt, 1ro6, 1rp7, 1rpf, 1rpj, 1rql, 1rr6, 1rtf, 1s19, 1s38, 1s39, 1s5z, 1s89, 1sqa, 1sqo, 1sqt, 1sr7, 1srg, 1stc, 1sv3, 1sw2, 1swg, 1swr, 1syh, 1syi, 1t4v, 1t5f, 1ta6, 1td7, 1thz, 1tjp, 1tkb, 1tng, 1tnh, 1tni, 1tow, 1tpw, 1tq4, 1trd, 1tsy, 1tt1, 1ttm, 1tx7, 1txr, 1u0g, 1u1w, 1u71, 1ua4, 1ugx, 1uho, 1ui0, 1uj5, 1ulb, 1uml, 1uou, 1upf, 1usi, 1usk, 1usn, 1utj, 1utm, 1utn, 1uto, 1uv6, 1uvt, 1uw6, 1uwf, 1uwt, 1uwu, 1uz1, 1uz4, 1v0k, 1v0l, 1v11, 1v1j, 1v1m, 1v2j, 1v2k, 1v2l, 1v2n, 1v2o, 1v2r, 1v2s, 1v2t, 1v2u, 1v2w, 1v48, 1v7a, 1vso, 1vzq, 1w11, 1w13, 1w2g, 1w3j, 1w3k, 1w4o, 1w4p, 1w4q, 1w7g, 1w96, 1wht, 1wm1, 1wn6, 1ws1, 1ws4, 1wvj, 1x1z, 1x38, 1x39, 1x8d, 1x8j, 1x8r, 1x8t, 1xap, 1xbo, 1xgi, 1xh4, 1xh9, 1xhy, 1xjd, 1xk9, 1xka, 1xm6, 1xoq, 1xow, 1xpz, 1xq0, 1xug, 1xws, 1y0l, 1y1z, 1y20, 1y3n, 1y3v, 1y3x, 1y6q, 1y6r, 1yc1, 1yc4, 1yda, 1ydb, 1ydd, 1ydr, 1yds, 1ydt, 1yei, 1yej, 1yfz, 1yp9, 1ype, 1ypg, 1ypj, 1yqj, 1yqy, 1yvf, 1yvm, 1ywr, 1z4o, 1z6s, 1z71, 1z95, 1z9g, 1z9y, 1zc9, 1zdp, 1zfq, 1zge, 1zgi, 1zhy, 1zoe, 1zog, 1zoh, 1zs0, 1zvx, 2a14, 2a5b, 2a8g, 2aac, 2ack, 2afw, 2afx, 2aj8, 2al5, 2ans, 2arm, 2ayr, 2b1g, 2b1i, 2b4l, 2b9a, 2baj, 2bal, 2bes, 2bfr, 2bmk, 2bo4, 2boh, 2boj, 2bok, 2brb, 2brm, 2bt9, 2buv, 2bvd, 2bvr, 2bys, 2bz6, 2bza, 2c1p, 2c3i, 2c3l, 2c97, 2ca8, 2cbj, 2cbu, 2cbv, 2cc7, 2ccb, 2ccc, 2ceq, 2cer, 2ces, 2cet, 2cex, 2cf8, 2cf9, 2cgf, 2cgr, 2cht, 2cle, 2clh, 2cli, 2clk, 2cn0, 2csn, 2ctc, 2d0k, 2d3u,

2doo, 2drc, 2dri, 2dw7, 2e1w, 2e27, 2e2p, 2e2r, 2e91, 2e92, 2e9u, 2epn, 2erz, 2ewa, 2ewb, 2ews, 2exm, 2ez7, 2f2h, 2f34, 2f35, 2f6t, 2f7i, 2f7o, 2f7p, 2f94, 2f9k, 2flr, 2fpz, 2fqo, 2fqt, 2fqw, 2fqx, 2fqy, 2fu8, 2fvd, 2fw6, 2fx6, 2fxs, 2fxv, 2fzc, 2fzg, 2fzk, 2g5u, 2gj5, 2gkl, 2gl0, 2glp, 2gss, 2gst, 2gsu, 2gvj, 2gvv, 2gyi, 2gz2, 2h15, 2h21, 2h3e, 2h4g, 2h4k, 2h4n, 2h6b, 2ha2, 2ha3, 2ha6, 2hah, 2haw, 2hb1, 2hjb, 2hl4, 2hmv, 2hnc, 2hoc, 2hu6, 2hxm, 2hzl, 2i19, 2i3i, 2i6b, 2i80, 2ihq, 2iko, 2isw, 2iuz, 2iwx, 2izl, 2j2u, 2j34, 2j47, 2j4g, 2j75, 2j77, 2j78, 2j79, 2j7b, 2j7d, 2j7e, 2j7f, 2j7g, 2j7h, 2j95, 2jdm, 2jdp, 2jds, 2jdu, 2jew, 2jf4, 2jfz, 2jg0, 2jgs, 2jh0, 2jh5, 2jh6, 2jiw, 2jjb, 2jkh, 2jkp, 2mas, 2mcp, 2nmx, 2nn1, 2nn7, 2nnd, 2nsj, 2nsl, 2nta, 2o0u, 2o4j, 2o4r, 2o4z, 2o8h, 2oag, 2oax, 2ogy, 2oi0, 2oi2, 2oiq, 2ojg, 2ojj, 2ole, 2on6, 2ot1, 2ovv, 2ovy, 2oxd, 2oxn, 2oxx, 2oxy, 2oym, 2p15, 2p16, 2p2a, 2p3i, 2p4s, 2p53, 2p7a, 2p7g, 2p7z, 2p95, 2pcp, 2pgz, 2pog, 2pou, 2pov, 2pow, 2pq9, 2pqb, 2pqc, 2pql, 2ptz, 2pu2, 2pvh, 2pvj, 2pvk, 2pvl, 2pvm, 2pwd, 2pwg, 2py4, 2q1q, 2q7q, 2q88, 2q89, 2q8h, 2q8z, 2qbq, 2qbr, 2qbs, 2qbu, 2qdt, 2qe4, 2qg0, 2qg2, 2qpq, 2qrk, 2qrl, 2qta, 2qtg, 2qtn, 2qtt, 2qu6, 2qw1, 2qwb, 2qwc, 2qwd, 2qwe, 2qwf, 2qzr, 2r2m, 2r2w, 2r58, 2r59, 2r5a, 2r9w, 2r9x, 2ra6, 2rcb, 2rcn, 2rd6, 2reg, 2rfh, 2ri9, 2rin, 2rk8, 2rka, 2rkd, 2rke, 2sim, 2tmn, 2usn, 2uwd, 2uwl, 2uwo, 2uwp, 2uxi, 2uy3, 2uy4, 2uy5, 2uyn, 2uyq, 2uz9, 2v00, 2v2c, 2v2h, 2v2q, 2v2v, 2v3d, 2v3u, 2v54, 2v57, 2v59, 2v77, 2v7a, 2v95, 2vba, 2vc9, 2ves, 2vfk, 2vhj, 2vjx, 2vk2, 2vk6, 2vl4, 2vmc, 2vmd, 2vmf, 2vnp, 2vnt, 2vot, 2vpn, 2vpo, 2vqt, 2vrj, 2vvc, 2vvn, 2vvs, 2vvu, 2vvv, 2vw1, 2vw2, 2vw5, 2vwl, 2vwn, 2vwo, 2vxn, 2vyt, 2w08, 2w26, 2w47, 2w4x, 2w66, 2w67, 2w8j, 2w8w, 2w8y, 2w9h, 2wb5, 2wbg, 2wc3, 2wc4, 2wca, 2we3, 2weg, 2weh, 2wej, 2wer, 2wf5, 2wgj, 2whp, 2wjg, 2wk6, 2wly, 2wlz, 2wn9, 2wnc, 2wnj, 2wor, 2wos, 2wq5, 2wr8, 2wtv, 2wuf, 2wvt, 2wvz, 2wyf, 2wyg, 2wyj, 2wzf, 2wzs, 2x00, 2x09, 2x0y, 2x2r, 2x4z, 2x7t, 2x7u, 2x8z, 2xab, 2xb8, 2xd9, 2xda, 2xde, 2xdk, 2xdl, 2xdx, 2xg9, 2xht, 2xib, 2xii, 2xj1, 2xj2, 2xj7, 2xjg, 2xjj, 2xjx, 2xm1, 2xm2, 2xmy, 2xn3, 2xn5, 2xnb, 2xp7, 2xpk, 2xxt, 2xys, 2y5f, 2y5g, 2y5h, 2y7x, 2y7z, 2y80, 2y81, 2y8c, 2ya7, 2ya8, 2yay, 2yaz, 2yb0, 2ydt, 2ydw, 2yek, 2yel, 2yfa, 2yfe, 2yfx, 2yhw, 2yi0, 2yk1, 2ylc, 2ymd, 2yme, 2yz3, 2z1w, 2z94, 2za0, 2za5, 2zb1, 2zc9, 2zcq, 2zcr, 2zcs, 2zda, 2zdk, 2zdl, 2zdm, 2zdn, 2zfp, 2zfs, 2zft, 2zgx, 2zjw, 2zkj, 2zmm, 2zq2, 2zwz, 2zx6, 2zx7, 2zx8, 2zxd, 2zxg, 2zy1, 2zz1, 2zz2, 3a1c, 3a1d, 3a6t, 3aas, 3aau, 3acl, 3acw, 3acx, 3ai8, 3aid, 3alt, 3ao2, 3ao4, 3ap4, 3aqt, 3arp, 3arq, 3arw, 3arx, 3axz, 3b1m, 3b24, 3b25, 3b26, 3b27, 3b3c, 3b3x, 3b4p, 3b50, 3b65, 3b66, 3b67, 3b68, 3b7i, 3b7j, 3b7r, 3b7u, 3b92, 3be9, 3bex, 3bft, 3bfu, 3bgq, 3bgs, 3bgz, 3bl0, 3bl1, 3bqc, 3bra, 3brn, 3bu1, 3buf, 3bug, 3buh, 3bxe, 3bxf, 3bxh, 3c2f, 3c2o, 3c2r, 3c39, 3c4h, 3c56, 3c79, 3c84, 3cf8, 3cfn, 3cft, 3cj2, 3cj4, 3cj5, 3ckz, 3cl0, 3cm2, 3cow, 3coy, 3coz, 3cs7, 3ctt, 3cyz, 3cz1, 3czv, 3d0b, 3d0e, 3d4y, 3d4z, 3d50, 3d51, 3d52, 3d6o, 3d6p, 3d78, 3d7z, 3d83, 3d8w, 3d8z, 3d9z, 3da9, 3daz, 3dbu, 3dc3, 3dcc, 3dd0, 3dd8, 3ddf, 3ddg, 3djo, 3djp, 3djv, 3djx, 3dnd, 3dne, 3dp4, 3dp9, 3dx1, 3dx2, 3dx3, 3dx4, 3dyo, 3e12, 3e3c, 3e5u, 3e92, 3e93, 3eb1, 3ebi, 3ebl, 3ebo, 3ebp, 3ed0, 3eft, 3egt, 3ehx, 3ehy, 3ejp, 3ejq, 3ejr, 3eko, 3ekr, 3elc, 3eqr, 3ewc, 3ewj, 3exe, 3exh, 3f15, 3f16, 3f17, 3f18, 3f19, 3f1a, 3f33, 3f34, 3f37, 3f3c, 3f5l, 3f68, 3f70, 3f78, 3f7g, 3f7h, 3f7i, 3f80, 3f8c, 3f8e, 3fat, 3fcq, 3fee, 3fh7, 3fhb, 3fj7, 3fjg, 3fl5, 3fqe, 3fql, 3fuc,

3fv1, 3fv2, 3fvk, 3fvn, 3fx6, 3g0e, 3g0i, 3g0w, 3g1d, 3g1v, 3g2y, 3g2z, 3g30, 3g31, 3g32, 3g34, 3g35, 3g3r, 3g5k, 3ga5, 3gba, 3gbb, 3gbe, 3gc4, 3gc5, 3gcp, 3gcs, 3gcu, 3gdt, 3ge7, 3gjw, 3gk1, 3gkz, 3gm0, 3gqz, 3gr2, 3gs6, 3gsm, 3gt9, 3gta, 3gtc, 3gv9, 3gvb, 3gvu, 3gy7, 3h1x, 3h30, 3h78, 3hb4, 3hcm, 3hek, 3hfb, 3hig, 3hkq, 3hkt, 3hku, 3hl5, 3hl7, 3hl8, 3hll, 3hmo, 3hmp, 3hs4, 3hub, 3huc, 3hww, 3hzm, 3i3b, 3i4b, 3i4y, 3i51, 3i5z, 3i60, 3i73, 3ibi, 3ibl, 3ibu, 3igp, 3ijh, 3ikd, 3ikg, 3imc, 3ime, 3iob, 3ioc, 3iod, 3ioe, 3iof, 3iog, 3iph, 3ipu, 3isj, 3iub, 3iue, 3ivc, 3ivx, 3iw6, 3jdw, 3jrs, 3jrx, 3juo, 3jup, 3jvr, 3jvs, 3jy0, 3jya, 3k1j, 3k4d, 3k5v, 3k8c, 3k8o, 3k8q, 3k97, 3k99, 3kdm, 3kek, 3kgp, 3kgq, 3kgt, 3kgu, 3kiv, 3kjd, 3kku, 3kmc, 3kmx, 3kmy, 3kqr, 3kr4, 3kr8, 3kv2, 3kwa, 3l0v, 3l3l, 3l3m, 3l4w, 3l4z, 3l59, 3ldp, 3ldq, 3le9, 3lea, 3lgs, 3liw, 3ljo, 3ljz, 3lk8, 3lp7, 3lpl, 3lq2, 3lvw, 3lxe, 3lxk, 3lzz, 3m1k, 3m3x, 3m3z, 3m40, 3m5e, 3m67, 3m6r, 3m96, 3mam, 3mdz, 3mf5, 3mfv, 3mfw, 3mhc, 3mhi, 3mhl, 3mhm, 3mho, 3mhw, 3mi2, 3miy, 3mjl, 3ml2, 3ml5, 3mmf, 3mna, 3mof, 3ms9, 3mss, 3muz, 3mv0, 3myg, 3mz6, 3mzc, 3n0n, 3n1c, 3n2p, 3n2v, 3n35, 3n3g, 3n3j, 3n4b, 3n76, 3n7a, 3n7o, 3n86, 3n8k, 3n9r, 3n9s, 3nb5, 3nee, 3nes, 3nex, 3ng4, 3ni5, 3nkk, 3nox, 3npc, 3nq3, 3nq9, 3nx7, 3nyd, 3nyx, 3nzk, 3o56, 3o5n, 3o75, 3o7u, 3o84, 3o8p, 3oaf, 3ocp, 3ocz, 3oe4, 3oe5, 3ohi, 3oil, 3oim, 3oku, 3ouj, 3ovn, 3oy0, 3oyq, 3oyw, 3ozg, 3ozj, 3ozp, 3ozr, 3ozs, 3ozt, 3p17, 3p2e, 3p3g, 3p3r, 3p3s, 3p3t, 3p4v, 3p5l, 3p5o, 3p7i, 3p8p, 3p8z, 3pb7, 3pb8, 3pb9, 3pbb, 3pce, 3pcf, 3pcg, 3pcj, 3pck, 3pcn, 3pd8, 3pd9, 3pe1, 3pe2, 3pfp, 3pgl, 3pgu, 3pn1, 3pn4, 3po1, 3po6, 3ppm, 3ppp, 3ppq, 3ppr, 3ps1, 3ptb, 3pwd, 3pwk, 3pyy, 3q2j, 3q44, 3q6w, 3qbc, 3qdd, 3qfy, 3qfz, 3qgw, 3qgy, 3qox, 3qps, 3qqa, 3qqs, 3qt6, 3qto, 3qtv, 3qwc, 3qx5, 3qxt, 3qxv, 3r16, 3r17, 3r1v, 3r24, 3r4m, 3r4n, 3r4p, 3r6u, 3r88, 3rbu, 3rdo, 3rdq, 3re4, 3rf4, 3rf5, 3rlb, 3rlp, 3rlq, 3rlr, 3rm4, 3rm9, 3roc, 3rr4, 3rsx, 3rt8, 3rtf, 3ru1, 3rv8, 3rwp, 3ryj, 3ryv, 3ryx, 3ryy, 3ryz, 3rz0, 3rz1, 3rz5, 3rz7, 3rz8, 3s0b, 3s0d, 3s0e, 3s2v, 3s5y, 3s6t, 3s71, 3s72, 3s73, 3s75, 3s76, 3s77, 3sfg, 3sha, 3shc, 3si3, 3si4, 3sk2, 3spf, 3sr4, 3sur, 3sus, 3sut, 3suu, 3suv, 3suw, 3sv2, 3sw8, 3sww, 3sxf, 3t01, 3t08, 3t09, 3t0b, 3t0d, 3t0x, 3t1a, 3t1m, 3t2q, 3t2w, 3t5u, 3t64, 3t70, 3t82, 3t84, 3t85, 3ta1, 3tao, 3tay, 3tb6, 3td4, 3tf6, 3tfn, 3tfp, 3tfu, 3th9, 3tif, 3tt4, 3ttm, 3tu7, 3tvc, 3twp, 3tz0, 3tza, 3tzm, 3u10, 3u5j, 3u5l, 3u81, 3u8j, 3u8k, 3u8l, 3u8n, 3u90, 3u92, 3u9q, 3ubd, 3ucj, 3udd, 3udh, 3uev, 3ug2, 3ui7, 3uil, 3uj9, 3ujc, 3ujd, 3umq, 3uo4, 3uod, 3up2, 3usx, 3uu1, 3uug, 3uuo, 3uw5, 3uxd, 3uxk, 3uxl, 3uz5, 3uzj, 3v2n, 3v51, 3v5p, 3v5t, 3v78, 3vbd, 3vd4, 3vdb, 3veh, 3vh9, 3vha, 3vhc, 3vhd, 3vhk, 3vvy, 3vw1, 3vw2, 3vx3, 3w07, 3w5n, 3w9k, 3w9r, 3wha, 3wjw, 3wmc, 3wtj, 3wtl, 3wtm, 3wtn, 3wto, 3wz6, 3wz7, 3wz8, 3wzn, 3x00, 3zbx, 3zc5, 3zcl, 3zdg, 3zdh, 3zdv, 3zhx, 3zi0, 3zi8, 3zll, 3zln, 3zm9, 3znr, 3zq9, 3zqe, 3zsq, 3zsx, 3zsy, 3zt2, 3zt3, 3zv7, 3zxz, 3zyf, 3zyu, 3zze, 456c, 4a4v, 4a4w, 4a6l, 4a6s, 4a7i, 4a95, 4ab9, 4aba, 4abb, 4abd, 4abe, 4abf, 4abg, 4abh, 4acc, 4aci, 4ad2, 4ad3, 4ad6, 4afg, 4ag8, 4agc, 4agl, 4agm, 4agn, 4agp, 4agq, 4ahr, 4ahs, 4ahu, 4ai5, 4aia, 4aj4, 4aje, 4aji, 4ajl, 4alx, 4aoi, 4ap7, 4aq4, 4aq6, 4aqh, 4ara, 4arb, 4asd, 4ase, 4asj, 4att, 4auj, 4av4, 4av5, 4avh, 4avj, 4avs, 4axd, 4ayp, 4ayq, 4ayu, 4az5, 4az6, 4azb, 4azc, 4azg, 4azi, 4b0b, 4b2i, 4b2l, 4b32, 4b33, 4b34, 4b35, 4b3c, 4b3d, 4b5d, 4b5s, 4b5t, 4b5w, 4b6o, 4b6p, 4b6r, 4b6s, 4b73, 4b74, 4b76, 4b7p, 4b7r, 4b9k, 4bah, 4bak, 4bam,

4ban, 4bao, 4baq, 4bb9, 4bc5, 4bck, 4bcm, 4bcn, 4bco, 4bcp, 4bcs, 4bf1, 4bf6, 4bi6, 4bi7, 4bj8, 4bks, 4bny, 4bqg, 4bqh, 4bqs, 4br3, 4bt3, 4bt4, 4btk, 4bup, 4buq, 4c1y, 4c52, 4c6u, 4c9x, 4cc5, 4cd0, 4cd4, 4cd5, 4cfl, 4cg9, 4cgi, 4cig, 4ciw, 4cj4, 4cjp, 4cjq, 4cjr, 4ck3, 4cl6, 4clj, 4cmo, 4cox, 4cp5, 4cpy, 4cpz, 4cr5, 4cr9, 4cra, 4crl, 4cs9, 4csd, 4css, 4cst, 4cu7, 4cu8, 4cwf, 4cwn, 4cwo, 4cwp, 4cwq, 4cwr, 4cws, 4cwt, 4d1j, 4d4d, 4d8z, 4da5, 4daf, 4db7, 4dbm, 4dcs, 4ddh, 4ddk, 4ddm, 4de0, 4de1, 4de2, 4de5, 4del, 4der, 4des, 4det, 4deu, 4dew, 4dff, 4dhl, 4dju, 4djv, 4djw, 4djx, 4djy, 4dko, 4dkp, 4dkq, 4dkr, 4dld, 4dmw, 4do4, 4do5, 4dst, 4dsu, 4duh, 4dv8, 4dzy, 4e1k, 4e3g, 4e4l, 4e4n, 4e5w, 4e6d, 4e6q, 4e70, 4e9u, 4ea2, 4eb8, 4ef6, 4efk, 4efs, 4egk, 4ehz, 4ei4, 4ej8, 4ejl, 4elg, 4elh, 4emf, 4en4, 4eo8, 4eoh, 4eor, 4epy, 4erf, 4euo, 4ew2, 4ew3, 4ewn, 4exs, 4f09, 4f1l, 4f2w, 4f39, 4f3c, 4f3k, 4f6u, 4f9u, 4fai, 4fcq, 4fev, 4few, 4ffs, 4fht, 4fk6, 4fl1, 4fl2, 4flp, 4fm7, 4fm8, 4fp1, 4fs4, 4fsl, 4fxp, 4fxq, 4fzj, 4g0p, 4g0q, 4g0y, 4g5f, 4g8m, 4g8n, 4g8v, 4g8y, 4g90, 4g95, 4ge1, 4gfm, 4gfo, 4gg7, 4ggz, 4ghi, 4gih, 4gii, 4gj2, 4gj3, 4gkh, 4gki, 4gkm, 4gny, 4gq4, 4gqp, 4gqq, 4gqr, 4gu6, 4gu9, 4gue, 4gzp, 4gzt, 4h3f, 4h3g, 4h3j, 4h42, 4h7q, 4h81, 4h85, 4ha5, 4hbm, 4hf4, 4hfp, 4hge, 4ht0, 4ht2, 4hu1, 4hw3, 4hwo, 4hwp, 4hws, 4hy1, 4hym, 4hzm, 4i3z, 4i54, 4i5c, 4i71, 4i72, 4i74, 4i7j, 4i7k, 4i7l, 4i7m, 4i7p, 4i8n, 4i8x, 4i9u, 4ibb, 4ibd, 4ibg, 4ido, 4ih3, 4ih5, 4ih6, 4ih7, 4iid, 4iie, 4iif, 4ij1, 4ipi, 4ipj, 4ipn, 4ish, 4isu, 4itp, 4iue, 4iuo, 4iva, 4ivb, 4ivc, 4ivd, 4iwz, 4j21, 4j22, 4j28, 4j3l, 4j7d, 4jal, 4je7, 4jfm, 4jfs, 4jh0, 4jia, 4jkw, 4jn2, 4jsa, 4jss, 4jsz, 4jwk, 4jx9, 4jxs, 4jyc, 4jym, 4jzi, 4k0o, 4k18, 4k3h, 4k3n, 4k4j, 4k55, 4k5p, 4k6i, 4k77, 4k7i, 4k7n, 4k7o, 4k9y, 4kcx, 4keq, 4kfq, 4kif, 4km0, 4km2, 4kmz, 4kn1, 4knj, 4knm, 4knn, 4kow, 4kp5, 4kp8, 4ks1, 4ks4, 4kwf, 4kwg, 4kwo, 4kxb, 4kxn, 4kyk, 4kz3, 4kz4, 4kz6, 4kz7, 4kzq, 4kzu, 4l19, 4l2l, 4l4v, 4l4z, 4l50, 4l51, 4l9i, 4lar, 4lbu, 4lch, 4leq, 4lhm, 4lhv, 4lj5, 4lj8, 4ljh, 4lk7, 4lko, 4lkq, 4llj, 4llk, 4llp, 4llx, 4lm0, 4lm1, 4lm2, 4lm3, 4lm4, 4loo, 4lov, 4lps, 4lrr, 4luz, 4lxz, 4ly1, 4lyw, 4lzr, 4lzs, 4m0e, 4m0f, 4m0r, 4m0y, 4m13, 4m14, 4m2r, 4m2u, 4m2v, 4m2w, 4m3p, 4m6u, 4m8e, 4m8h, 4mc6, 4mgd, 4mhz, 4mjp, 4mme, 4mmm, 4mmp, 4mnp, 4mo4, 4mo8, 4mpn, 4mq6, 4mr3, 4mr6, 4mre, 4mrg, 4mrw, 4mrz, 4msa, 4msc, 4msn, 4mss, 4muf, 4mul, 4n07, 4n5d, 4n6g, 4n6z, 4n7m, 4n7u, 4n8q, 4n9c, 4na9, 4nbk, 4nbl, 4nbn, 4ndu, 4ngn, 4nh7, 4nh8, 4nj9, 4nja, 4nl1, 4non, 4np2, 4np3, 4np9, 4nra, 4nuc, 4nue, 4nvp, 4nwc, 4nxu, 4nxv, 4nyf, 4o04, 4o05, 4o07, 4o09, 4o0b, 4o0x, 4o0y, 4o2b, 4o2p, 4o3f, 4o61, 4oag, 4oak, 4oc0, 4oc2, 4ocq, 4oct, 4og3, 4og4, 4ogj, 4oiv, 4oks, 4or4, 4or6, 4ovf, 4ovg, 4ovh, 4owm, 4owv, 4ozj, 4p3h, 4p58, 4p5d, 4p5z, 4p6c, 4p6w, 4pb2, 4pcs, 4pee, 4pf5, 4pft, 4pfu, 4phu, 4pin, 4pmm, 4pnu, 4poh, 4poj, 4pop, 4pow, 4pox, 4pp0, 4pp3, 4pp5, 4pqa, 4psb, 4pum, 4pv5, 4pvx, 4pvy, 4q08, 4q09, 4q0k, 4q19, 4q3t, 4q3u, 4q46, 4q4o, 4q4p, 4q4q, 4q4r, 4q4s, 4q6d, 4q6e, 4q7p, 4q7s, 4q7v, 4q7w, 4q81, 4q83, 4q87, 4q8x, 4q8y, 4q90, 4q93, 4q99, 4q9o, 4q9y, 4qac, 4qb3, 4qd6, 4qdk, 4qem, 4qer, 4qev, 4qew, 4qf7, 4qf8, 4qf9, 4qgd, 4qge, 4qjw, 4qjx, 4ql1, 4qnb, 4qp2, 4qpd, 4qsu, 4qsv, 4qxo, 4qy3, 4qyy, 4r06, 4r0a, 4r3w, 4r59, 4r5a, 4r5b, 4r5t, 4r74, 4r75, 4r76, 4ra1, 4rak, 4rd0, 4rd3, 4rd6, 4rdn, 4re2, 4re4, 4rfd, 4rfm, 4rfr, 4riu, 4riv, 4rj8, 4rlt, 4rlu, 4rlw, 4rpn, 4rpo, 4rqv, 4rra, 4rrf, 4rrg, 4rsk, 4rux, 4ruy, 4ruz, 4rvr, 4rwj, 4s1g, 4sga, 4std, 4tim, 4tjz, 4tkb, 4tkh, 4tpw, 4tqn, 4trc, 4tte, 4tu4, 4tun, 4twp, 4ty6, 4ty7, 4tz2,

4u0f, 4u0w, 4u1b, 4u43, 4u54, 4u5o, 4u69, 4u6c, 4u6w, 4u6z, 4u70, 4u71, 4u73, 4ual, 4ucc, 4ufh, 4ufi, 4ufj, 4ufk, 4ufl, 4ufm, 4uin, 4uj1, 4uj2, 4uja, 4ujb, 4uma, 4umb, 4umc, 4und, 4unp, 4uof, 4uoh, 4up5, 4ury, 4urz, 4uye, 4uyf, 4v24, 4v27, 4w52, 4w9c, 4w9d, 4w9f, 4w9h, 4w9i, 4wa9, 4whs, 4wiv, 4wkb, 4wkn, 4wko, 4wkp, 4wop, 4wov, 4wrb, 4x24, 4x48, 4x50, 4x5p, 4x5q, 4x5r, 4x5z, 4x8u, 4xaq, 4xar, 4xas, 4xip, 4xiq, 4xir, 4xit, 4xo8, 4xoc, 4xoe, 4xt2, 4xu3, 4xxh, 4xy8, 4xya, 4y0a, 4y2q, 4y3j, 4y3y, 4y4j, 4y59, 4y79, 4yc0, 4yes, 4ygf, 4yhm, 4yho, 4yk0, 4ykk, 4ymb, 4ymg, 4ymh, 4yml, 4ynb, 4yo8, 4yrd, 4ysl, 4ytc, 4yth, 4yx4, 4yxi, 4yyt, 4yzu, 4z07, 4z0k, 4z0q, 4z1e, 4z1j, 4z1k, 4z2b, 4z83, 4z84, 4z93, 4zae, 4zbf, 4zbi, 4zcs, 4zei, 4zek, 4zgk, 4zji, 4zme, 4zo5, 4zow, 4zt8, 4zvi, 4zw5, 4zw6, 4zw7, 4zw8, 4zwz, 4zx0, 4zx1, 4zx3, 4zx4, 4zzd, 4zzx, 4zzy, 4zzz, 5a5q, 5a6k, 5a6x, 5a7b, 5a81, 5aa9, 5aan, 5aba, 5acy, 5afv, 5ahw, 5am6, 5am7, 5amd, 5amg, 5aml, 5ant, 5anu, 5anv, 5aoj, 5aol, 5aqz, 5aut, 5avf, 5ayt, 5b25, 5b5f, 5b5g, 5boj, 5btx, 5bv3, 5bw4, 5bwc, 5byi, 5c1w, 5c28, 5c2a, 5c2h, 5c2o, 5c3p, 5c5t, 5c8n, 5cap, 5caq, 5cas, 5cau, 5cbm, 5cbr, 5cbs, 5cc2, 5cep, 5ceq, 5chk, 5cj6, 5cjf, 5cks, 5cp5, 5cp9, 5cqt, 5cs3, 5cs6, 5cso, 5csp, 5cst, 5ct2, 5cu4, 5d0c, 5d0r, 5d1r, 5d21, 5d24, 5d25, 5d26, 5d3h, 5d3j, 5d3n, 5d3p, 5d3t, 5d45, 5d47, 5d48, 5dbm, 5dex, 5dey, 5dfp, 5dh4, 5dh5, 5dkn, 5dnu, 5dpx, 5dqe, 5dqf, 5drr, 5duw, 5dw2, 5dwr, 5dx4, 5dxt, 5dyo, 5e13, 5e1s, 5e28, 5e2k, 5e2l, 5e2r, 5e2s, 5e73, 5e74, 5e7n, 5ect, 5edb, 5edc, 5edd, 5edl, 5eei, 5eek, 5een, 5ef7, 5efh, 5efj, 5egu, 5eh5, 5eh7, 5ehr, 5ehv, 5ehw, 5ei3, 5eij, 5eis, 5ekm, 5el9, 5elw, 5en3, 5eng, 5ep7, 5eq1, 5eqy, 5er4, 5etb, 5etj, 5eu1, 5ev8, 5evb, 5evd, 5evk, 5evz, 5ew0, 5ewa, 5ewk, 5ewy, 5exl, 5exm, 5eyr, 5f08, 5f0f, 5f1h, 5f1r, 5f25, 5f2p, 5f2r, 5f5z, 5f61, 5f74, 5f8y, 5f9b, 5fbi, 5fck, 5fcz, 5fdc, 5fdi, 5fe6, 5fe9, 5fh7, 5fh8, 5fhn, 5fho, 5fl4, 5fl5, 5fl6, 5flo, 5fls, 5flt, 5fnc, 5fnd, 5fnf, 5fng, 5fnr, 5fns, 5fnt, 5fog, 5fol, 5fou, 5fox, 5fpk, 5fs5, 5fsc, 5fsn, 5fso, 5fsx, 5ftg, 5fut, 5fyx, 5g17, 5g1a, 5g1z, 5g2b, 5g2g, 5g45, 5g46, 5g4m, 5g4n, 5g4o, 5g5f, 5g5v, 5gj9, 5gmh, 5gmn, 5gsa, 5h1t, 5h1v, 5h5f, 5h85, 5h8e, 5h9r, 5ha1, 5hbs, 5hct, 5hcv, 5his, 5hjq, 5hrv, 5hrw, 5hrx, 5htz, 5hva, 5hvs, 5hvt, 5hwu, 5hwv, 5hz5, 5hz6, 5hz8, 5hz9, 5i1q, 5i29, 5i2f, 5i3w, 5i7x, 5i7y, 5i80, 5i88, 5i8g, 5i9y, 5i9z, 5ia1, 5ia2, 5ia3, 5igm, 5ih9, 5ihh, 5ii2, 5ijr, 5ikb, 5ime, 5ioz, 5ipj, 5ito, 5itp, 5ivc, 5ive, 5ivv, 5ivy, 5iwg, 5ix0, 5iyy, 5j0d, 5j1r, 5j1x, 5j20, 5j2x, 5j3l, 5j6a, 5j6l, 5j6m, 5j6n, 5j7q, 5j7w, 5j82, 5j86, 5j8m, 5j8u, 5j8z, 5j9x, 5jgi, 5jgq, 5jhb, 5jhk, 5ji8, 5jq5, 5js3, 5jsq, 5jss, 5jt9, 5jvi, 5jxn, 5k03, 5k0m, 5k1d, 5k1f, 5k8s, 5k9w, 5ka1, 5ka7, 5ka9, 5kab, 5kad, 5kat, 5kax, 5kbe, 5kby, 5kcb, 5kh3, 5khm, 5kly, 5km9, 5kma, 5ko1, 5ko5, 5kva, 5kz0, 5l2s, 5l3a, 5l4i, 5l4j, 5l4m, 5l7e, 5l7h, 5l8a, 5l8c, 5l9i, 5ld8, 5ldm, 5ljq, 5ljt, 5llc, 5lle, 5llg, 5llh, 5lli, 5llo, 5llp, 5lne, 5lny, 5lom, 5lsg, 5ltn, 5lud, 5lvd, 5lvl, 5lvq, 5lvr, 5lwd, 5lwm, 5lyr, 5lz4, 5lz5, 5lz7, 5m17, 5m5d, 5m7s, 5m7u, 5m9w, 5meh, 5mek, 5mg2, 5mge, 5mgf, 5mgj, 5mgk, 5mjn, 5mkr, 5mks, 5mlj, 5mme, 5mmg, 5mn1, 5mnn, 5mo8, 5mod, 5mpk, 5mpn, 5mpz, 5mqe, 5mrm, 5mro, 5mrp, 5msb, 5mwh, 5mwp, 5mwy, 5mxf, 5my8, 5mz8, 5n0d, 5n0e, 5n0f, 5n17, 5n18, 5n1r, 5n1s, 5n1z, 5n24, 5n25, 5n34, 5n3v, 5n6s, 5n84, 5n93, 5n9r, 5nap, 5nau, 5nbw, 5ndf, 5ne5, 5nea, 5neb, 5nee, 5ngz, 5nih, 5njz, 5nk2, 5nk3, 5nk4, 5nk7, 5nk9, 5nka, 5nkb, 5nkc, 5nkd, 5nkg, 5nki, 5nlk, 5nn5, 5nn6, 5nvv, 5nvw, 5nvx, 5nw1, 5nw2, 5nw7, 5nwe, 5nxg, 5nxi, 5nxo, 5nxp, 5nxw, 5nya, 5nyh,

5nz4, 5nze, 5nzf, 5nzn, 5o07, 5o1d, 5o1f, 5o1h, 5o2d, 5o4f, 5o5a, 5odx, 5oei, 5oh2, 5oh3, 5oh4, 5oh7, 5oh9, 5oha, 5oku, 5om2, 5om3, 5op4, 5op5, 5oqu, 5org, 5orh, 5orj, 5ork, 5orv, 5orw, 5os2, 5os4, 5os5, 5os7, 5os8, 5ose, 5osl, 5oss, 5ot8, 5ot9, 5ota, 5otc, 5otr, 5otz, 5ouh, 5ov8, 5ovr, 5ovx, 5owl, 5qa8, 5qay, 5std, 5sym, 5sz0, 5sz1, 5sz2, 5sz3, 5sz4, 5sz6, 5sz7, 5t19, 5t7s, 5t8o, 5tb6, 5tbm, 5tcy, 5tfx, 5th4, 5ti0, 5tpx, 5tt3, 5tuo, 5tuz, 5twj, 5txy, 5ty9, 5u0d, 5u0e, 5u0f, 5u0g, 5u0w, 5u0y, 5u0z, 5u11, 5u13, 5u14, 5u28, 5u49, 5u4b, 5u4d, 5u8c, 5uc4, 5ueu, 5uez, 5uf0, 5ufp, 5ufr, 5ufs, 5uk8, 5ula, 5uln, 5ulp, 5umx, 5umy, 5uoo, 5upe, 5upf, 5upj, 5ut6, 5uv2, 5v79, 5v7a, 5v82, 5var, 5vb5, 5vb6, 5vb7, 5vcv, 5vcw, 5vd0, 5vd1, 5vd2, 5vgy, 5vih, 5vij, 5vja, 5vl2, 5vm0, 5vo1, 5vsf, 5vyy, 5w1e, 5wa8, 5wa9, 5wal, 5wbm, 5wbo, 5wcm, 5we9, 5wex, 5wgp, 5wl0, 5wp5, 5wqc, 5wuk, 5wyx, 5wyz, 5x62, 5xg5, 5xmx, 5xo7, 5xpi, 5xsr, 5xva, 5xvg, 5y12, 5y13, 5y8y, 5y94, 5ya5, 5yas, 5yfs, 5yft, 5yh8, 5yhe, 5yhg, 5yj8, 5yl2, 5yz2, 5z5f, 5z7b, 5z7j, 5z99, 5za7, 5za8, 5za9, 5zae, 5zaf, 5zag, 5zaj, 5zc5, 5zkc, 5zo8, 6aqs, 6ayi, 6ayo, 6ayq, 6b1k, 6b4d, 6b4l, 6b59, 6b7a, 6b7b, 6b96, 6b97, 6b98, 6bdy, 6bm5, 6bm6, 6c7q, 6c7x, 6cbf, 6cbg, 6ce6, 6ced, 6chp, 6cjv, 6ckr, 6cks, 6ckw, 6cn5, 6cpa, 6csp, 6csq, 6csr, 6css, 6cvf, 6cwh, 6cwn, 6czb, 6czc, 6d2o, 6d50, 6d55, 6d56, 6d5e, 6d5g, 6d5j, 6d9x, 6dai, 6dak, 6dar, 6dq4, 6ebe, 6ecz, 6eed, 6eeo, 6ei5, 6eif, 6eij, 6eiq, 6eir, 6eis, 6eiz, 6ej2, 6ej3, 6el5, 6eln, 6elo, 6elp, 6en5, 6eog, 6ep4, 6epa, 6epz, 6eq8, 6eqp, 6equ, 6euw, 6eux, 6evr, 6ex1, 6exi, 6exs, 6ey8, 6ey9, 6eya, 6eyb, 6eyt, 6f1j, 6f1n, 6f20, 6f28, 6f3b, 6f90, 6f92, 6f9g, 6fa4, 6faa, 6faf, 6fba, 6fe0, 6fe1, 6fhk, 6fhq, 6fmj, 6fnf, 6fng, 6fni, 6fnj, 6fnq, 6ftz, 6fuh, 6fui, 6fuj, 6fyz, 6g0z, 6g2m, 6g34, 6g36, 6g37, 6g38, 6g39, 6g3q, 6g3v, 6g9i, 6g9u, 6gfs, 6gfz, 6ghh, 6gjj, 6gjl, 6gjm, 6gjr, 6gl9, 6glb, 6gnm, 6gnp, 6gnr, 6gnw, 6gon, 6got, 6guc, 6gue, 6guh, 6guk, 6gvz, 6gw4, 6gzd, 6gzm, 6h29, 6h2z, 6h33, 6h34, 6h36, 6h37, 6h38, 6h5x, 6h8s, 6hd6, 6hke, 6hlx, 6hly, 6hmg, 6hpw, 6hqy, 6hrq, 6hsh, 6ht1, 6iiu, 6rnt, 6std, 6upj, 7std, 7upj, 8a3h, 8cpa, 966c

6.6.2 Test set:

1a4r, 1a69, 1bhx, 1f8c, 1fhd, 1g45*, 1g53*, 1ghy, 1ghz, 1gyx, 1j37*, 1jqy, 1lgw, 1lpk, 1oss, 1oyt, 1p19, 1pxo, 1tmn, 1tom, 1ucn*, 1utl, 1v16, 1vfn, 1wcq, 1yq7, 2a5c, 2ax9, 2azr, 2bet, 2bq7, 2d3z, 2e7f, 2e94, 2f1g, 2hzy, 2j27, 2j62, 2j94, 2jke*, 2pbw, 2pu1*, 2q38, 2q8m, 2qm9, 2std, 2v58, 2vb8, 2vo4, 2vo5, 2vuk, 2vzr*, 2weo*, 2wky, 2x91, 2y82, 2ya6, 2yi7, 2yix, 2yki, 2ypi, 3a1e, 3ao5, 3b4f*, 3b5r, 3bbf, 3bxg, 3c52, 3cd5, 3djq, 3e5a*, 3ebh, 3fed, 3ffp, 3fvl, 3gst, 3gy2, 3gy3, 3gy4, 3hk1, 3hp9, 3ip9, 3iww, 3k37, 3k5x, 3lka, 3myq*, 3neo, 3owj, 3p58, 3s78*, 3std, 3t3u, 3ueu, 3uw4, 3v7x*, 3vd9, 3zj6, 4arw, 4b7j, 4bkt, 4bs0, 4bt5, 4c1u, 4cga, 4dsy, 4e0x, 4ek9, 4eo6, 4f9w, 4f9y, 4g0z, 4gbd, 4ibc, 4iic, 4isi, 4j93, 4k0y, 4kao, 4kiu, 4kn0, 4kni, 4kyh, 4m12, 4mhy, 4muv, 4myd, 4n9a, 4o9v, 4oma, 4p6x, 4qtl*, 4r73, 4rfc*, 4rqk, 4rr6, 4y5d, 4yha, 5aoi, 5c1m, 5cy9, 5d3l, 5dlx, 5dq8, 5eh8, 5eqe, 5eqp, 5fe7, 5flq, 5gja, 5h1u, 5i3a, 5i3v, 5ie1, 5j27, 5j64, 5jox, 5jxq, 5l7g, 5nk6, 5ny3*, 5oq8, 5ost, 5qal, 5sz5*, 5t8p, 5tya, 5u12, 5ucj, 5w44*, 6ayr, 6c7w*, 6d5h, 6ekq, 6f05, 6fgg, 6fnr, 6fv4*, 6g35, 6g3a, 6gji, 6gla, 6htg, 6mjf

*Incompatible with NNScore due to unsupported atom type(s)