A Study on Protein-Ligand Binding Affinity Prediction with Machine Learning Methods

P072, P475, P890, P928

MSc in Statistical Science, HT 2025
Word Count: 2497

1 Introduction

This project investigates molecular binding affinity to a protein target using 1,157 samples, each represented by two feature sets: `fps` and `embed`. The `fps` dataset has 1,024 binary indicators representing the Morgan fingerprint, while the `embed` dataset has 224 feature variables extracted via Convolutional Neutral Network (CNN).

Both feature sets are divided into three subsets with no missing values: training set, public test set, and private test set, with their sizes summarised in Table 1.

Datasets	**fps**	**embed**
`training_set`	(975,1,024)	(975,224)
`public_test_set`	(109,1,024)	(109,224)
`private_test_set`	(73,1,024)	(73,224)

Table 1: Summary of dataset sizes for the `fps` feature set with 1,024 binary variables and the `embed` feature set with 224 continuous variables, across the training, public test, and private test subsets. Each entry is shown as (number of samples, number of variables).

2 Exploratory Data Analysis

2.1 Characteristics of Predictors and Response

2.1.1 Predictors

For `fps`. Figure 1 shows that `X_fps_train` and `X_fps_public_test` datasets are highly sparse, where large blank areas suggest a great proportion of zero entries. Despite this sparsity, certain substructures appear in all molecules [1], while some others are entirely absent [2] across both `fps` training and public test sets.

For `embed`. Similar to `fps`, several columns in `embed` datasets are entirely zero [3] across both training and public test sets. Unlike `fps`, `embed` features vary in scale, where some columns

[1] For example, the 357^{th} and 379^{th} fingerprints, which corresponds to the column indices 356 and 378.
[2] For example, fingerprints with column indices 321 and 331.
[3] Column Indices: 1, 5, 6, 14, 16, 19, 20, 21, 22, 23, 29, 32, 36, 39, 42, 45, 46, 55, 58, 60, 62, 68, 69, 73, 77, 78, 80, 85, 87, 88, 92, 96, 99, 106, 108, 111, 115, 116, 117, 122, 129, 130, 131, 135, 136, 137, 147, 149, 152, 157, 159.

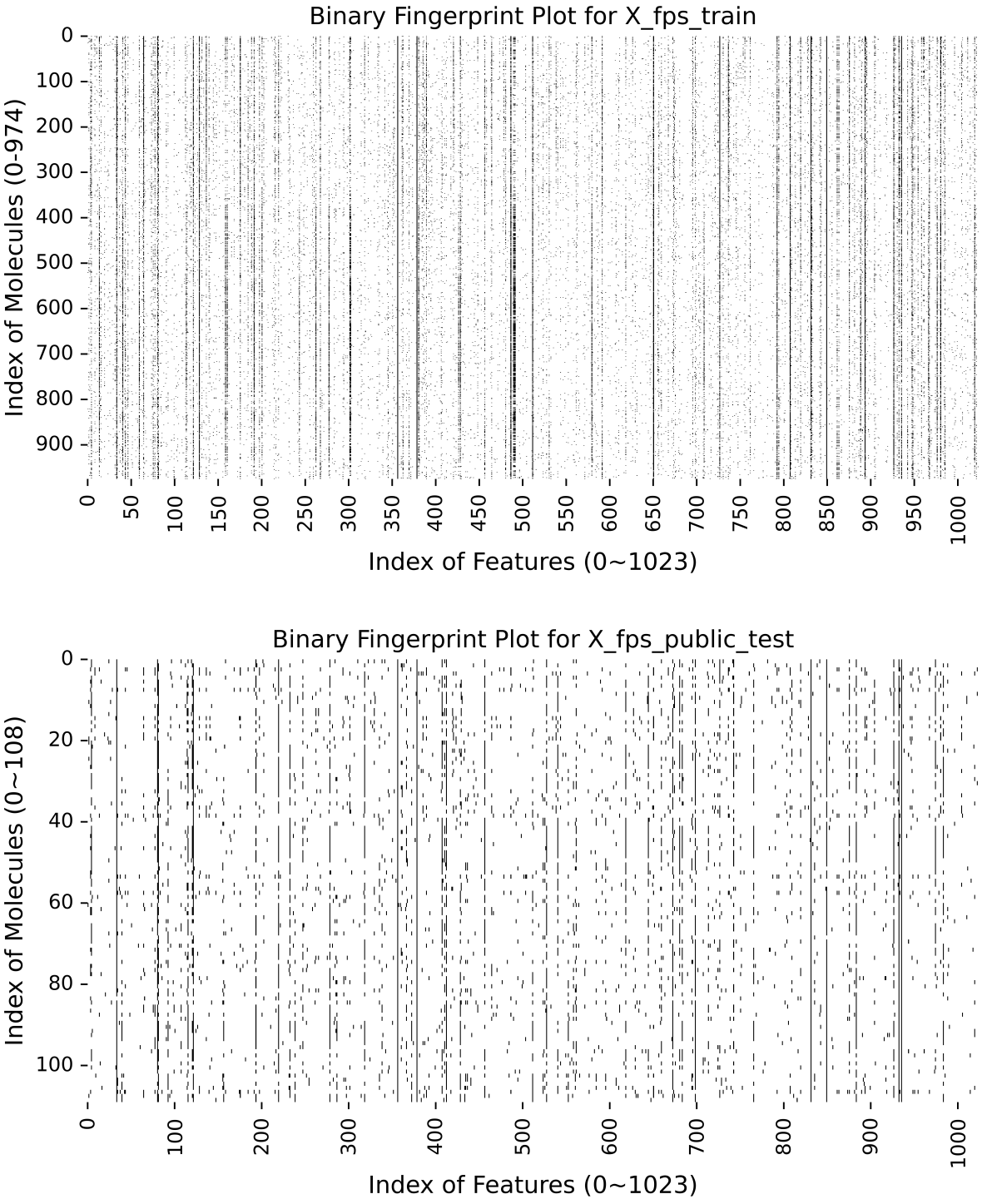

Figure 1: Binary Fingerprint plots for the `X_fps_train` (above) and `X_fps_public_test` (below). Black dots indicate the presence of specific molecular substructures.

have much larger ranges than others (Figure 2). This suggests the need for standardisation in the pre-processing stage for some models.

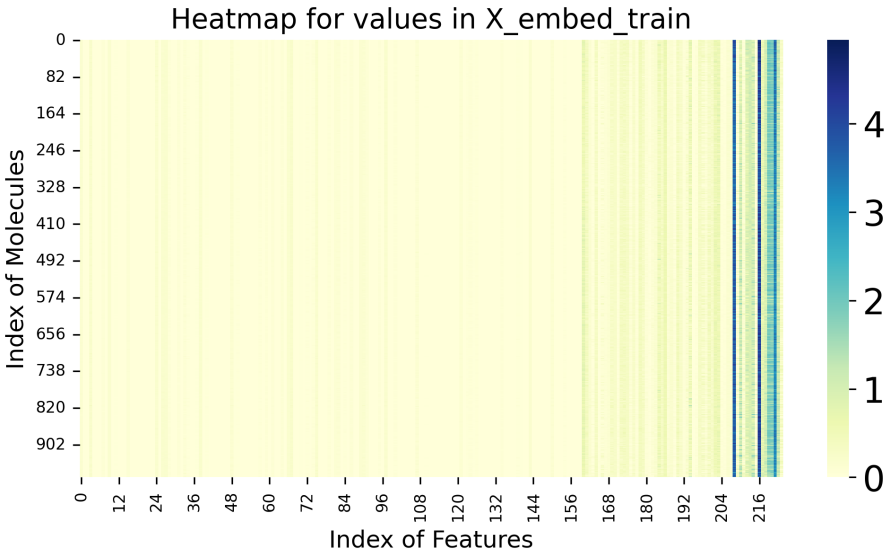

Figure 2: Heatmap of feature values in `X_embed_train`, showing variation in scales across different feature indices. Darker shades indicate higher values.

2.1.2 Response

Histograms in Figure 3 present a higher proportion of large binding affinity molecules in training set (`y_train`), compared to public test set (`y_public_test`).

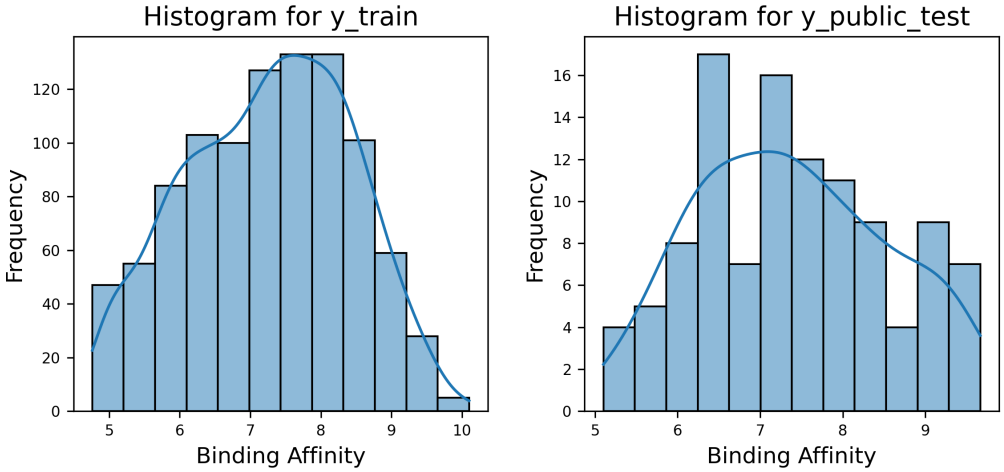

Figure 3: Histograms of binding affinity values in `y_train` (Left) and `y_public_test` (Right). While the `y_train` exhibits a roughly bell-shaped distribution, the public test set shows a less bell-shaped with a higher proportion of molecules having large binding affinity values.

2.2 Distribution of Training and Public Test Sets

To investigate potential distributional shift, `X_fps_combined` and `X_embed_combined` were formed by row-wise concatenation of training and public test datasets for `fps` and `embed` features respectively.

t-Distributed Stochastic Neighbor Embedding (t-SNE) was applied to project the combined datasets into 2D for visualisation. Figure 4 shows an obvious separation between public test (red) and training set (blue) in both `fps` and `embed` features. The red dots form a regional cluster that only partially overlap with the blue dots, which suggests the training and public test observations may come from different distributions.

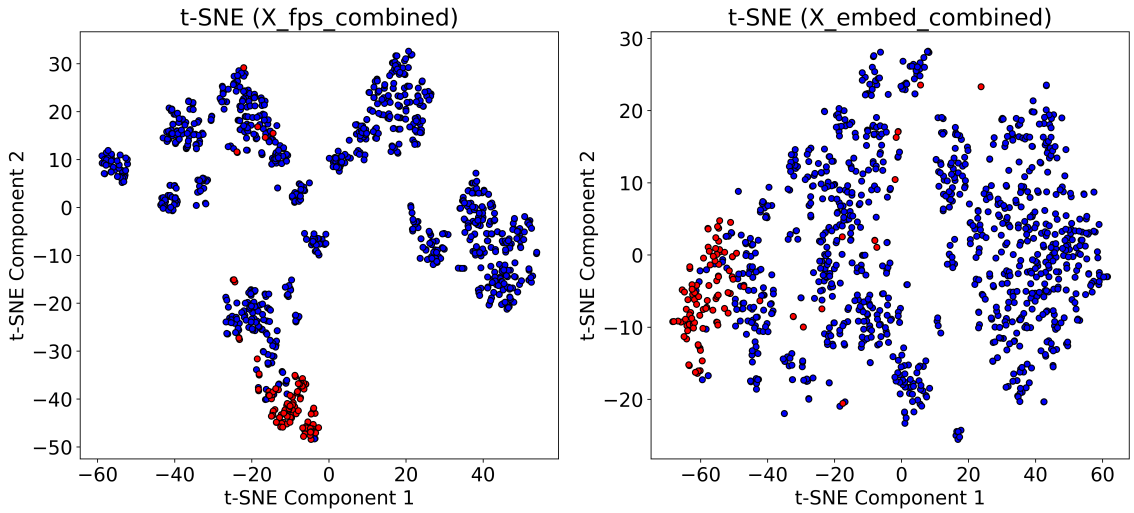

Figure 4: t-SNE visualisations of `X_fps_combined` (left) and `X_embed_combined` (right), formed by combining training and public test sets. Blue and red dots indicate samples from the training and public test sets, respectively.

Additionally, a Random Forest Classifier could classify the combined dataset well by achieving F1-score of 0.94 for `fps` and 0.96 for `embed`. This indicates a clear distribution discrepancy.

3 Fingerprints and Embed Data Modelling

To reduce implicit ordering influence and ensure more reliable cross-validation, we **shuffled** `fps` and `embed` training sets before train-validation split.

We considered parametric models, including Linear Regression and Elastic Net, but they performed poorly in this case. Therefore, we focus on three more effective approaches: **Support Vector Regression (SVR)**, **Tree-based Models** and **Neural Network (NN) Models**. These models are evaluated on both `fps` and `embed` features, based on their model characteristics and performance.

3.1 Support Vector Regression

SVR fits a function such that most predicted values fall within an ϵ-margin of tolerance, penalising only those outside this range (Figure 5). This ϵ-insensitive loss makes SVR robust to noise and effective for capturing complex, non-linear relationships in data.

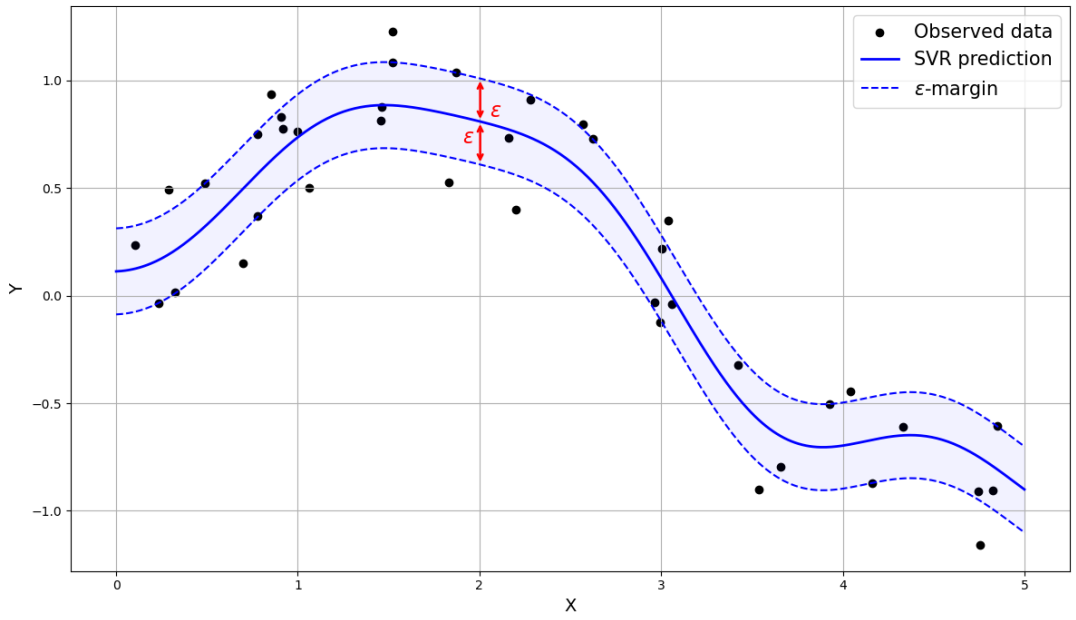

Figure 5: An Illustrative example of SVR with non-linear fit. The dashed blue lines represent the ϵ-margin tube, with ϵ shown in red.

Before training SVR on `embed` data, `MinMax` scaling was applied to rescale all features to $[0, 1]$ range, as `embed` training set shows varying feature scales (Figure 2). This is necessary because SVR relies on distance-based kernels and is sensitive to feature scales. Scaling ensures a balanced contribution from all features, which removes bias caused by scale differences.

3.1.1 Dimensionality Reduction Strategy

As both `fps` and `embed` datasets are high-dimensional, two dimensionality reduction methods are applied to each dataset: (i) Principal Component Analysis (PCA) that preserves variance across components, and (ii) Non-negative Matrix Factorisation (NMF) extracts additive parts-based representation.

However, the effectiveness of these methods varies by feature type. According to Table 2, based on the Test Mean Square error (MSE), SVR performed best on `fps` without dimensionality reduction, whereas `embed` benefited most from PCA. This is because:

- For binary and sparse `fps` features, dimensionality reduction can eliminate informative signals with low-variance. The reduced feature space becomes denser but less interpretable, degrading SVR performance.
- For continuous `embed` features, their high density after scaling makes PCA effective in extracting the most informative components, enabling SVR to generalise better.

3.1.2 SVR Hyperparameter Tuning

For `fps`, SVR was trained on original features without the dimensionality reduction method (none + SVR). For `embed`, PCA was applied prior to SVR to compress the feature space (PCA + SVR).

Feature	Method	MSE
`fps`	none	**0.8240**
	PCA	0.9504
	NMF	1.2921
`embed`	none	0.8202
	PCA	**0.6896**
	NMF	0.7788

Table 2: Test MSE for SVR with a non-linear RBF kernel evaluated across feature types and dimensionality reduction methods. Best MSE for each feature type is in bold.

SVR hyperparameters for both features were optimised through 5-fold grid search Cross-validation, with selection based on Cross-validated MSE. These hyperparameters includes:

1. regularisation parameter C, balancing model complexity and training error;
2. margin width ϵ, which defines the tolerance around target error;
3. types of kernel and kernel-specific parameters, i.e. γ for RBF kernel and the degree for polynomial kernel.

3.1.3 SVR Model Selection Discussion

The hyperparameters were fine-tuned, with values of C and ϵ chosen to reflect the appropriate levels of regularisation and tolerance for each feature set:

- For `fps`, the best SVR configuration uses an RBF kernel with `gamma = 'scale'`, $C = 10.0$, and $\epsilon = 0.01$.
  - The higher C and smaller ϵ reflect a stricter fit, suitable for sparse binary features where precise matching is beneficial.
- For `embed`, the optimal SVR model (with PCA reduced to 130 components) also uses an RBF kernel with `gamma = 'scale'`, $C = 1.0$, and $\epsilon = 0.1$.
  - The lower C and larger ϵ allow a smoother fit with greater tolerance for small errors, which is suitable for continuous features. As PCA further simplifies the input, it reduced the need for strong regularisation.

3.2 Tree-based Models: Random Forest & XGBoost

Tree-based models were used for their ability to perform implicitly feature selection via split criteria, which help to identify relevant feature subsets. They also adapt to different feature types of `fps` and `embed`.

As single decision trees prone to overfitting when too deep and may underfit when overly constrained, we focused on more robust ensemble methods: **Random Forest** (reduces variance through bagging) and **XGBoost** (improves performance through boosting).

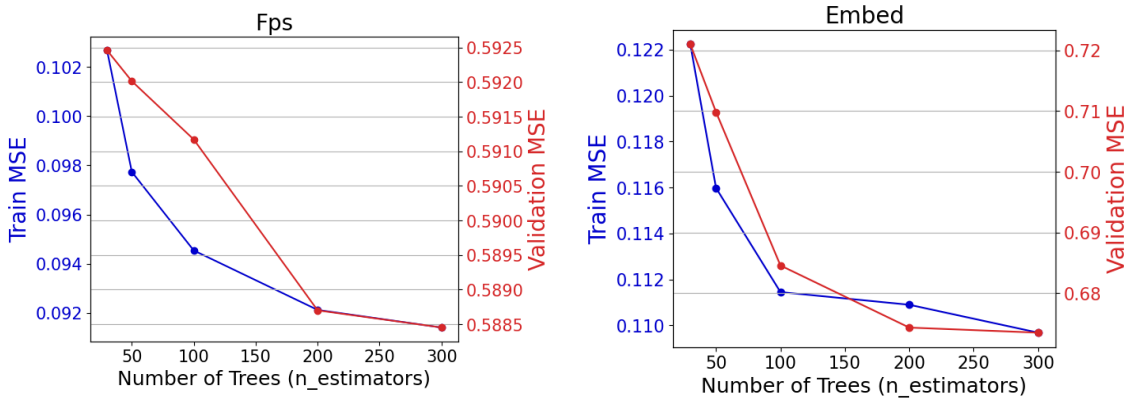

Figure 6: The exploration of `n_estimators` in Random Forest: The change of train MSE and validation MSE against different number of trees (`n_estimators`).

3.2.1 Random Forest

Initial Model Exploration

We started with a moderate tree depth (`max_depth = 20`) and examined how risk varied with the number of trees (`n_estimators`), as presented in Figure 6. As out-of-bag risk can be unstable with small and varying samples, we used 4:1 train-validation split to monitor train and validation MSE. Based on this, `n_estimators` was set to 200 for both datasets to balance computational cost and generalisability in the following grid-search hyperparameter tuning.

Hyperparameter Tuning

Following the same approach as SVR, grid search based on validation MSE was performed on two key tree-related parameters:

1. `max_depth`: the maximum depth of each tree, which controls model complexity;
2. `max_features`: the proportion of features considered at each split, influencing the diversity among trees in the forest.

3.2.2 XGBoost

Initial Model Exploration

We performed 5-fold cross-validation with a large initial number of boosting rounds `n_estimators` = 1000. Figure 7 shows both training and validation MSE decreased steadily but improved little after approximately 500 rounds. Hence, `n_estimators` = 500 was chosen to balance predictive performance and computational cost.

Hyperparameter Tuning

Same as the previous models, XGBoost hyperparameters were selected based on cross-validation MSE, focusing on the following parameters:

1. `max_depth`: controls the model complexity of each tree;
2. `colsample_bytree`: the fraction of features randomly selected for each tree, promoting diversity;

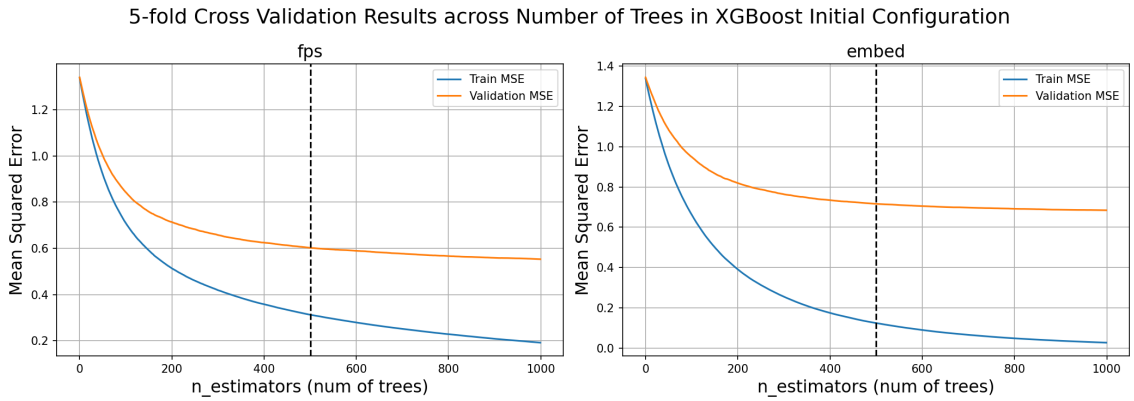

Figure 7: The exploration for XGBoost initial configuration. The figure shows in both datasets, the change of train MSE and validation MSE as `n_estimators` increases.

3. `learning_rate` for boosting;
4. `subsample`: the proportion of training samples used per tree; lower values increase randomness and reduce overfitting;
5. `gamma`: a post-pruning regularisation term that sets the minimum loss reduction required to split a leaf node.

First four parameters were determined through grid search to obtain the best cross validation model. Then, we tuned the regularisation parameter `gamma` by referring to the distribution of split gains, followed by another grid search to find its optimal value.

3.2.3 Tree-based Model Selection Discussion

The best ***Random Forest*** configurations were identified as follows, balancing model complexity, generalisability, and overfitting reduction across both feature sets:

- For `fps`: `n_estimators` = 200, `max_depth` = 25, `max_features` = 0.8
  - The moderate tree depth helps prevent overfitting to trivial substructures presented in sparse binary features.
  - A higher feature fraction at each split ensures rare but informative substructures are included in decision-making.
- For `embed`: `n_estimators` = 200, `max_depth` = 30, `max_features` = 0.5
  - Deeper trees capture the non-linear relationships in the complex molecule–protein interaction space.
  - A lower feature fraction per split helps reduce overfitting from correlated continuous features.

The optimal ***XGBoost*** configurations were determined as follows:

- Common hyperparameters: `n_estimators` = 500, `max_depth` = 15, `learning_rate` = 0.01, `subsample` = 0.5, for both `fps` and `embed`.
  - The moderate tree depth limits the model complexity.
  - Subsampling half the observations promotes generalisation through randomness.

- `gamma`: For `fps`: 0.0688; For `embed`: 0.0088.
  - `gamma` was set to lower quantile of split-gains of each dataset's trees to prune low-gain branches, which mitigates overfitting.
- `colsample_bytree`: For `fps`: 0.8; For `embed`: 0.5.
  - Similar to `max_features` in Random Forest, a higher feature fraction in `fps` ensures that uncommon substructures are covered by most trees, while a lower fraction in `embed` is selected to account for the correlation between features.

3.3 Neural Network Models

NNs are powerful non-parametric models known for their flexibility in learning complex, non-linear relationships. However, since NNs have a strong tendency to overfit, careful model design is essential when modelling both `fps` and `embed` datasets.

Before training, the dataset was randomly split into training and validation sets using a 4:1 ratio. The training set was further divided into 25 batches, each containing 32 samples. Model optimisation used the Adam optimiser, with MSE as the loss function. All NN models were implemented with PyTorch, using Metal Performance Shaders (MPS) backend for GPU acceleration. Training was conducted on Apple M1 Pro GPUs with 32GB of unified memory.

3.3.1 `fps` Features: Model Design and Model Training

Due to high sparsity in `fps` feature, NNs suffer from capturing meaningful patterns and may lead to vanishing gradients. Therefore, we used an embedding technique to reduce dimensionality and extract meaningful information, assigning each of the 1,024 features a latent vector of size 256. We then computed the average latent vector across the features present in each molecule, forming a condensed representation. This approach enables the model to focus specifically on existing features while ignoring the rest. Subsequently, the computed latent representation was passed through a Feed-Forward Network (FFN) to generate predictions. Based on empirical experiments, we built an FFN architecture with three hidden layers of 1,024, 512, and 256 dimensions, respectively. To prevent overfitting, a dropout rate of 0.5 was applied to each hidden layer.

Training proceeded for 500 epochs, starting with an initial learning rate of 0.0005, which linearly decayed to zero after 400 epochs for precise tuning. The trained model achieved a validation performance of MSE $= 0.6034 \pm 0.0314$ and MAE $= 0.6008 \pm 0.0223$.

3.3.2 `embed` Features: Model Design and Model Training

As `embed` feature is naturally suitable for prediction with FFNs, we adopted the same FFN architecture previously used for `fps` features.

Training was conducted for 1,000 epochs with initial learning rate of 0.0001, which linearly decayed to zero by 700 epochs. The trained model achieved a validation performance of MSE $= 0.7637 \pm 0.0335$ and MAE $= 0.6727 \pm 0.0180$.

3.4 Model Comparison

We evaluated SVR, tree-based methods, and NNs separately on public test sets of `fps` and `embed`, and discuss their relative performance below.

3.4.1 `fps` Feature Modelling

Model	MSE	MAE	Runtime (s)
SVR	**0.7204**	**0.6781**	**0.8332 ± 0.0124**
Random Forest	0.7562 ± 0.0157	0.7282 ± 0.0050	3.6615 ± 0.1855
XGBoost	0.7462 ± 0.0187	0.7087 ± 0.0076	1.4157 ± 0.1827
Embedding+FFN	1.0464 ± 0.1657	0.8086 ± 0.0646	116.3843 ± 0.7501

Table 3: Benchmarking results on `fps` feature for four selected models. The reported metrics represent the mean and standard deviation over five runs. For SVR, MSE and MAE results remain constant across repetitions due to its deterministic nature.

Both SVR and Tree-based models achieved strong predictive performance on `fps` features. SVR was proven as the best-performing model, due to (i) its ability to model complex, non-linear relationships via RBF kernel, (ii) robustness against irrelevant features, and (iii) minimal need for explicit feature selection. By comparison, tree-based models performed well but less impressively than SVR.

In contrast, FFN was notably worse. Ablation studies revealed that the embedding layer was crucial for stable training, but suffered from generalising to the public test distribution.

3.4.2 `embed` Feature Modelling

Among all three models, SVR with PCA was the best-performing model for `embed` features. PCA first reduces dimensionality and noise, allowing SVR's non-linear modelling to operate more effectively on limited data. This has improved accuracy and prevented overfitting even with low regularisation parameter C.

Tree-based models struggle to balance precision and generalisation in this case, since accurately modelling continuous response demands numerous splits and may overfit or underfit. Although XGBoost's residual-based boosting improves over Random Forest, neither method matches SVR's strong generalisability.

As `embed` features are extracted via a CNN, a FFN can effectively capture their underlying patterns. It outperformed Random Forest and XGBoost, but still lagged behind SVR. The models has shown a good generalisability, with similar training and validation results.

Model	MSE	MAE	Runtime(s)
PCA + SVR	**0.6826**	**0.6611**	**0.2050 ± 0.0332**
Random Forest	0.8452 ± 0.0139	0.7561 ± 0.0064	6.8194 ± 0.4224
XGBoost	0.8125 ± 0.0150	0.7372 ± 0.0059	16.4061 ± 0.4567
FFN	0.7505 ± 0.0259	0.6847 ± 0.0124	93.9869 ± 2.2636

Table 4: Benchmarking results on `embed` feature for four selected models. The reported metrics represent the mean and standard deviation over five runs. For SVR, MSE and MAE results remain constant across repetitions due to its deterministic nature.

3.5 Interpretability & Computational Cost

Both SVR and tree-based models offer some interpretability: SVR (Figure 8) and XGBoost (Figure 9) utilises SHAP values to express features' influence on the response bind-

ing affinity, and Random Forests use permutation-based feature importances (Figure 10). Conversely, neural network preserves low interpretability because of its complex, "black box" architecture, making it difficult to discern how individual features drive the model's decisions.

Among all the models, SVR proves the most computationally efficient on both datasets, since it relies on only a subset of training examples as the support vectors to define its decision function. Random Forest also demonstrates good efficiency but lags behind SVR, while XGBoost is efficient for sparse `fps` data but much slower for `embed` due to its sequential tree-building process in continuous space (without GPU-acceleration in our report). Finally, FFN exhibits the highest computational cost amongst all models across both datasets.

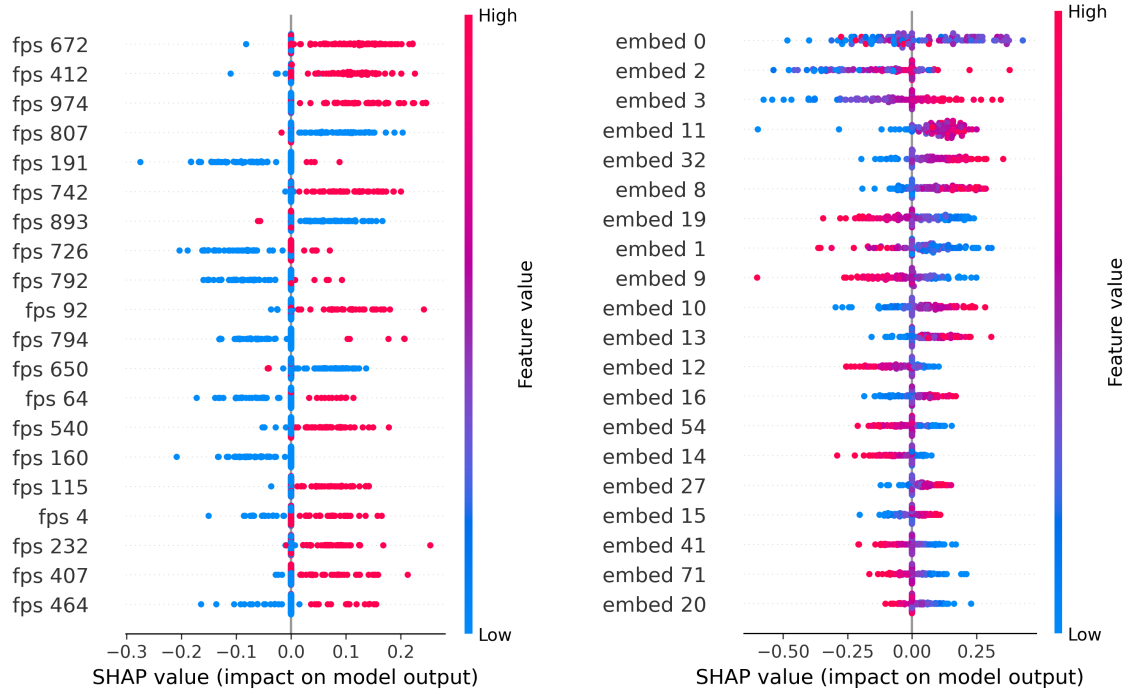

Figure 8: SVR's SHAP plots for `fps` and `embed` datasets (using KernelExplainer). We only adopted 100 training samples (molecules) as background for kernel SHAP value calculation efficiency. Features are ranked by SHAP importance (mean absolute SHAP value). For each variable, every molecule appears as one point in the plot. A positive SHAP value (plotted to the right) indicates that this feature increases the binding affinity for that molecule, while a negative value (left) lowers it. The color of points (red for high value, blue for low value) reveals how feature value correlates with the direction and magnitude of its effect on prediction.

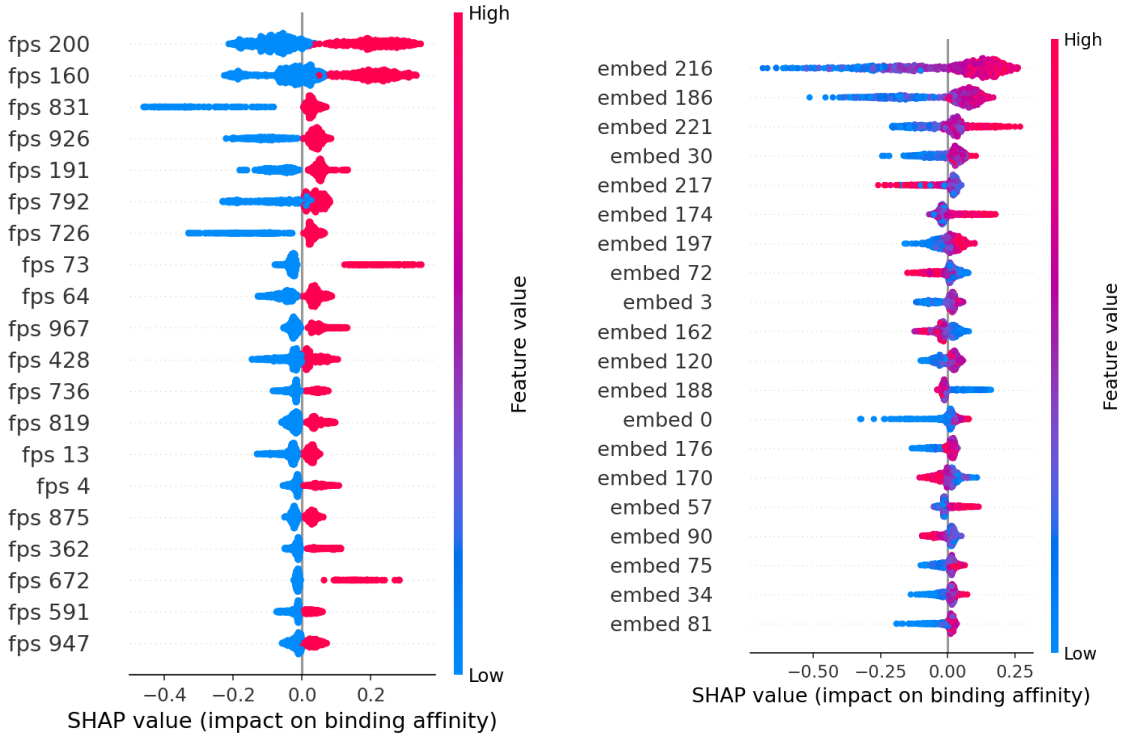

Figure 9: XGBoost's tree SHAP (SHapley Additive exPlanations) plots for both `fps` and `embed` datasets (using TreeExplainer).

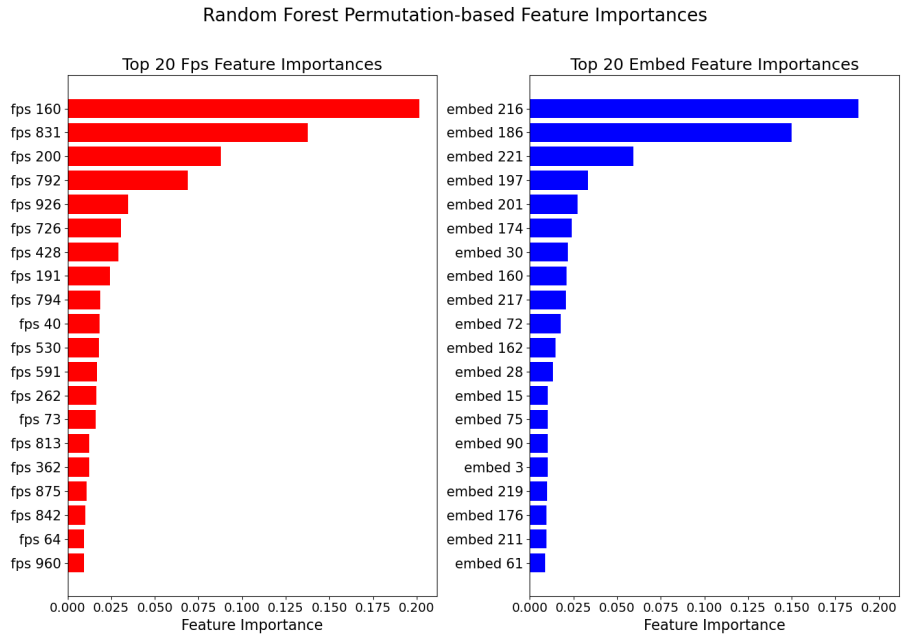

Figure 10: Random Forest provides Permutation-based feature importance plots for both `fps` and `embed` datasets. Top 20 features are visualised.

4 Final Model

For the final model, we combine both `fps` and `embed` features into one dataset, evaluate on public test set, and select the best-performing model to predict on private test set.

4.1 Joint Dataset via Horizontal Concatenation

The joint dataset is created by horizontally concatenating `fps` and `embed` features, **applying scaling for SVR but not for tree-based models**.

4.1.1 SVR

Although both sets were scaled to $[0, 1]$, their variance structures differ: binary `fps` features have limited variance, while continuous `embed` features exhibit wider variance. Variance-based projection in PCA can suppress informative patterns in low-variance binary features. Therefore, SVR without dimensionality reduction is chosen as the final model, with grid-search optimised hyperparameters: $C = 10$, $\epsilon = 0.01$, `kernel = 'rbf'`, `gamma = 'scale'`.

4.1.2 Tree-based Models

Since decision trees rely on relative comparisons for splitting, scaling is unnecessary and does not affect their ability to partition the combined feature space.

4.2 Feature Fusion via Weighted Addition Fusion

We developed a Weighted Addition Fusion (WAF) module to integrate information from both features. The `fps` and `embed` features were first mapped to 256 dimensions using embedding and linear transformation, resulting in z_{fps} and z_{embed}. Then, it was passed to the WAF module for data fusion.

The resulting fused vector is then processed using the same FFN architecture described in Section 3. Training was conducted for 700 epochs, starting with a learning rate 0.0001, which decayed linearly to zero after 500 epochs. The trained model achieved a validation performance of MSE $= 0.6915 \pm 0.1014$ and MAE $= 0.6667 \pm 0.0654$.

4.3 Model Comparison

SVR consistently performs best, as it is suitable for high-dimensional, mixed-type data. The C parameter provides reliable regularisation, preventing overfitting. Compared to using either feature set alone, SVR achieves even better performance on the joint dataset. This is because combining binary and continuous features improves similarity computation within RBF kernel, allowing SVR to operate in a more expressive hypothesis space and capture complex structure–property relationships more effectively. Particularly, Figure 12 shows all top 20 important features are from the `fps` data, highlighting `fps`'s important contribution to the joint dataset.

For Random Forest, permutation-based feature importance indicates that `embed` features dominate predictions in the joint dataset, thereby diluting the signal from `fps` and reducing overall prediction performance. In contrast, XGBoost uses the complementary information from both feature sets to outperform single-dataset XGBoost models, although it still failed to surpass SVR's predictive performance.

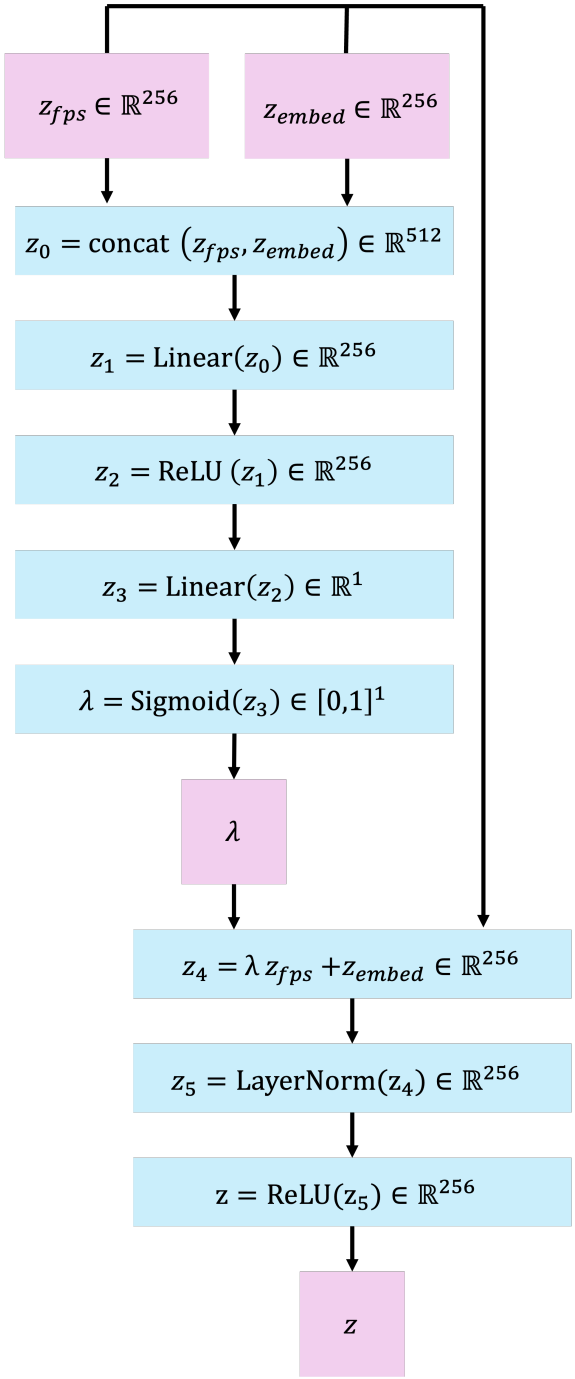

Figure 11: Architecture for WAF module. Based on the findings in Section 3, where `embed` features outperformed `fps` in prediction tasks, the WAF module was designed to retain all information from `embed` while selectively involving only weighted information from `fps`.

For NNs, ablation studies suggest WAF module enhances fusion over direct concatenation, the overall model performance is weaker than traditional models due to weak generalisabilty in this case.

Model	MSE	MAE	Runtime(s)
SVR	**0.632**	**0.6393**	**0.9942 ± 0.025**
Random Forest	0.8124 ± 0.0120	0.7400 ± 0.0076	4.2911 ± 1.9102
XGBoost	0.7076 ± 0.0295	0.6928 ± 0.0150	7.1431 ± 0.9770
WAF+FFN	0.8981 ± 0.0734	0.7676 ± 0.0202	300.14664 ± 3.3384

Table 5: Benchmarking results on joint feature for four selected models. The reported metrics represent the mean and standard deviation over five runs. For SVR, MSE and MAE results remain constant across repetitions due to its deterministic nature.

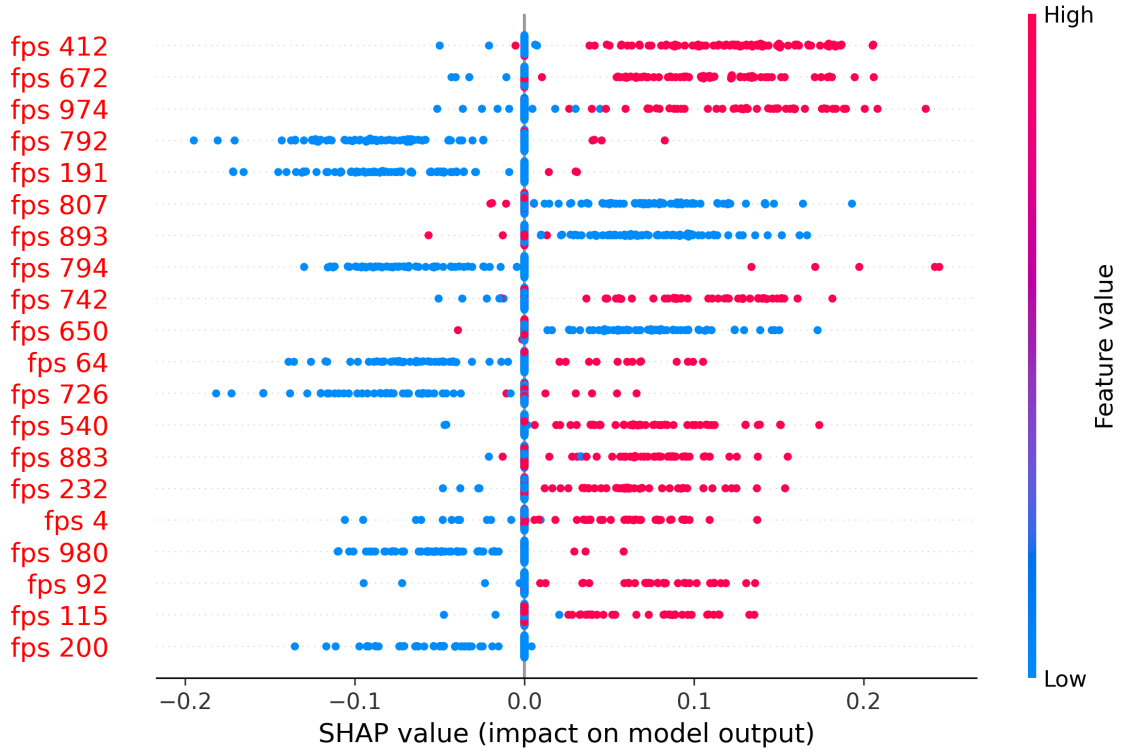

Figure 12: SHAP plot for SVR model with joint dataset.

5 Conclusion

Under all modelling scenarios, SVR achieved the lowest MSE and MAE on public test sets, indicating its strong prediction accuracy and generalisation ability. Consequently, SVR is selected as the final model for predicting binding affinity on `y_private_test`, given its superior performance with joint `fps` and `embed` features.

6 Appendix

All codes for this project are available on GitHub. The codes provided below include only the implementation of the final predictor, performance evaluation, and predictions on the private test set.

```

import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.preprocessing import MinMaxScaler

###############
# Load Dataset
###############

# Load fps feature data
X_fps_train = pd.read_csv(
    './data/X_fps_train.csv', index_col=0
)
X_fps_public_test = pd.read_csv(
    './data/X_fps_public_test.csv', index_col=0
)
X_fps_private_test = pd.read_csv(
    './data/X_fps_private_test.csv', index_col=0
)

# Load embed feature data
X_embed_train = pd.read_csv(
    './data/X_embed_train.csv', index_col=0
)
X_embed_public_test = pd.read_csv(
    './data/X_embed_public_test.csv', index_col=0
)
X_embed_private_test = pd.read_csv(
    './data/X_embed_private_test.csv', index_col=0
)

# Load response data
y_train = pd.read_csv(
        './data/y_train.csv', index_col=0
    ).to_numpy() # outputs of the training set
y_public_test = pd.read_csv(
        './data/y_public_test.csv', index_col=0
    ).to_numpy() # outputs of the training set

np.random.seed(2025)
random_idx = np.random.permutation(y_train.shape[0])

X_fps_train = X_fps_train.iloc[random_idx].reset_index(drop=True)
X_embed_train = X_embed_train.iloc[random_idx].reset_index(drop=True)
y_train = y_train[random_idx]

```

```
##############################
# Preprocess the embed feature
##############################

# Forces range between 0 and 1
scaler_embed = MinMaxScaler(feature_range=(0, 1))
X_embed_train = scaler_embed.fit_transform(X_embed_train)
X_embed_public_test = scaler_embed.transform(X_embed_public_test)
X_embed_private_test = scaler_embed.transform(X_embed_private_test)

##################################
# Join data for two feature types
##################################

# Step 1: Concatenate fps and embed features
X_train_joint = pd.DataFrame(np.hstack((X_fps_train, X_embed_train)))
X_public_test_joint = pd.DataFrame(np.hstack((X_fps_public_test,
↪  X_embed_public_test)))
X_private_test_joint = pd.DataFrame(
    np.hstack((X_fps_private_test, X_embed_private_test))
)

# Step 2: Ensure y_train is in 1D format
y_train = y_train.flatten()
y_public_test = y_public_test.flatten()

#########################
# Fit SVR with grid search
#########################

# Set up grid for grid search
param_grid_svr = {
    'C': [1, 10, 100],  # Regularization parameter
    'epsilon': [0.01, 0.1],  # Tolerance margin
    'kernel': ['rbf', 'poly'],  # Kernel types
    'degree': [2, 3, 4], # Only used for polynomial kernel
    'gamma': ['scale','auto', 0.1, 1]
}

# Fit SVR with grid search
grid_search = GridSearchCV(
    SVR(), param_grid_svr,
    cv=5, scoring='neg_mean_squared_error', n_jobs=-1, verbose=5
)
grid_search.fit(X_train_joint, y_train)

##################################
# Validation and test performance
##################################

# Derive the best model and its details
best_svr_model = grid_search.best_estimator_
best_svr_params = grid_search.best_params_
best_validation_mse = -grid_search.best_score_

print("Best Parameters:", best_svr_params)
```

```
105 print(f"Best Validation MSE: {best_validation_mse:.4f}.")
106
107 # Test on the public test set
108 y_pred = best_svr_model.predict(X_public_test_joint)
109
110 mse_test = mean_squared_error(y_public_test, y_pred)
111 mae_test = mean_absolute_error(y_public_test, y_pred)
112
113 print(f"Test Error - MSE: {mse_test:.4f}, MAE: {mae_test:.4f}.")
114
115 ##########################
116 # Predict on private test
117 ##########################
118
119 # Compute predictions on the private test set
120 y_pred = best_svr_model.predict(X_private_test_joint)
121
122 # Export the predictions on the test data in csv format
123 prediction = pd.DataFrame(y_pred, columns=['Prediction'])
124 prediction.index.name='Id'
125 prediction.to_csv('myprediction.csv') # export to csv file
126
127
```