Freie Universität Berlin

GraphSynergy:
A Comprehensive Evaluation of Performance, and Graph-Based Learning in Drug Synergy Prediction

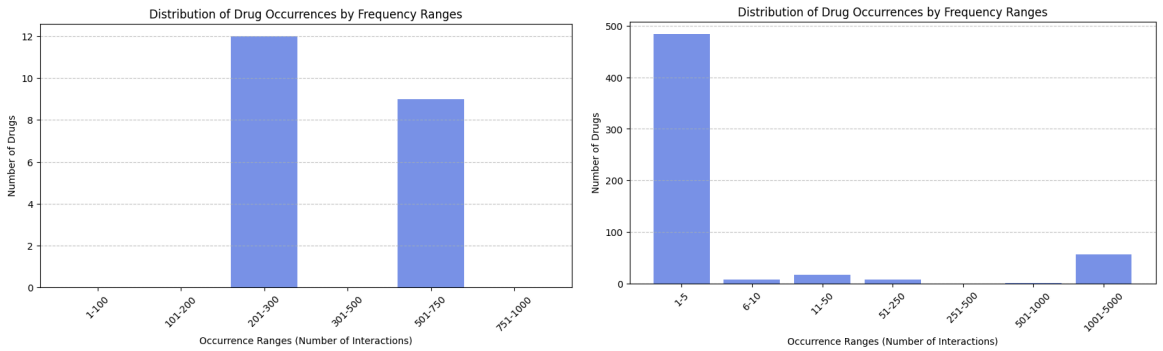
Authors:
Andreas Klæboe,
Kerem Aras,
Julian Hesse,
Felix Trau

Introduction

Analyzing The Data

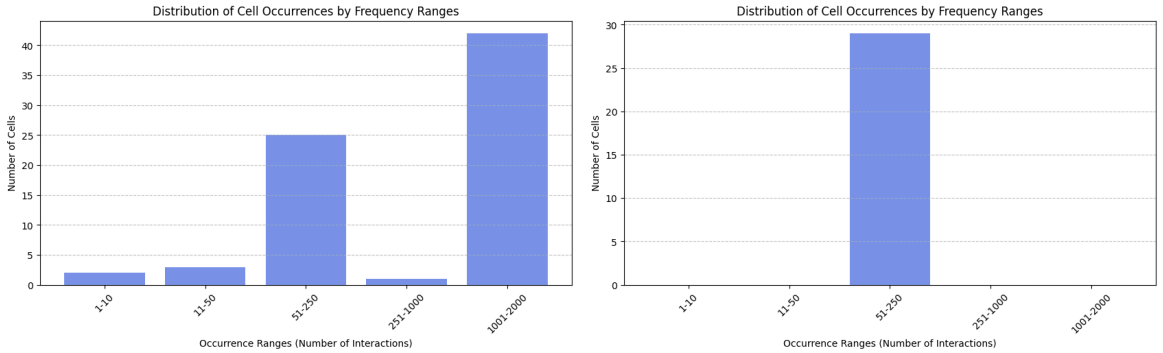
Overview

We analyzed two datasets from the paper: **DrugCombDB** and **Oncology Screen**. DrugCombDB contains **70,000 samples**, while Oncology Screen has **4,000 samples**.



Single Drug Occurence: Left DrugCombDB, Right Oncology Screen

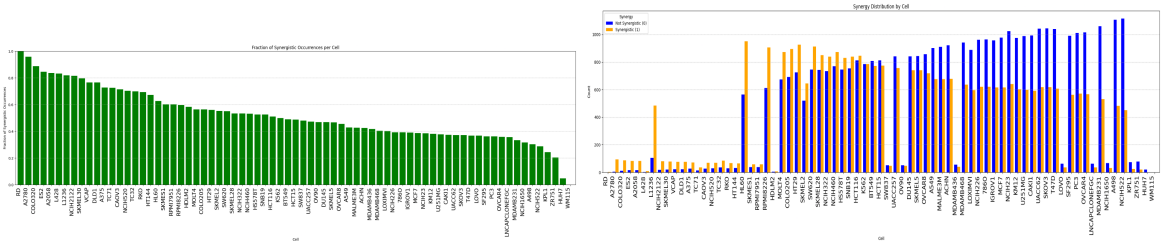
In DrugCombDB, a large portion of single drugs appear fewer than 10 times. However, the number of drugs with a sufficient sample size (more than 100 occurrences) is still higher than in Oncology Screen.



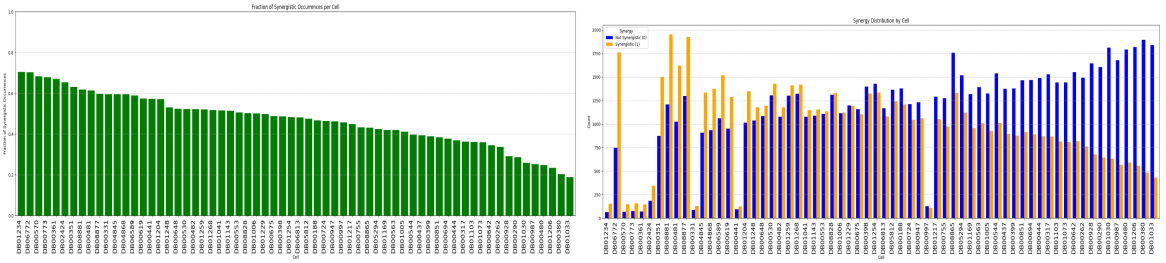
Single Cell Occurence: Left DrugCombDB, Right Oncology Screen

For single cell lines, there are far fewer unique occurrences, making the data more consistent and easier to work with.

Oncology Screen shows a better distribution of cell lines, but the total amount of usable data is much lower compared to DrugCombDB.



Label imbalance: DrugCombDB without rarest drugs (<10) - relative on the left, absolute on the right

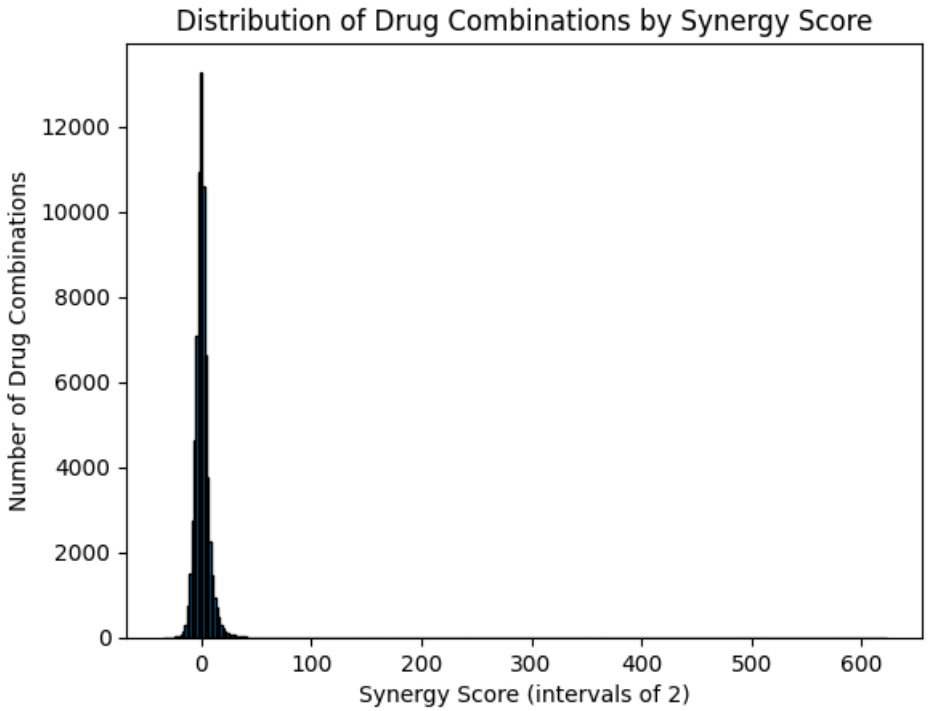


Label imbalance: Oncology Screen - relative on the left, absolute on the right

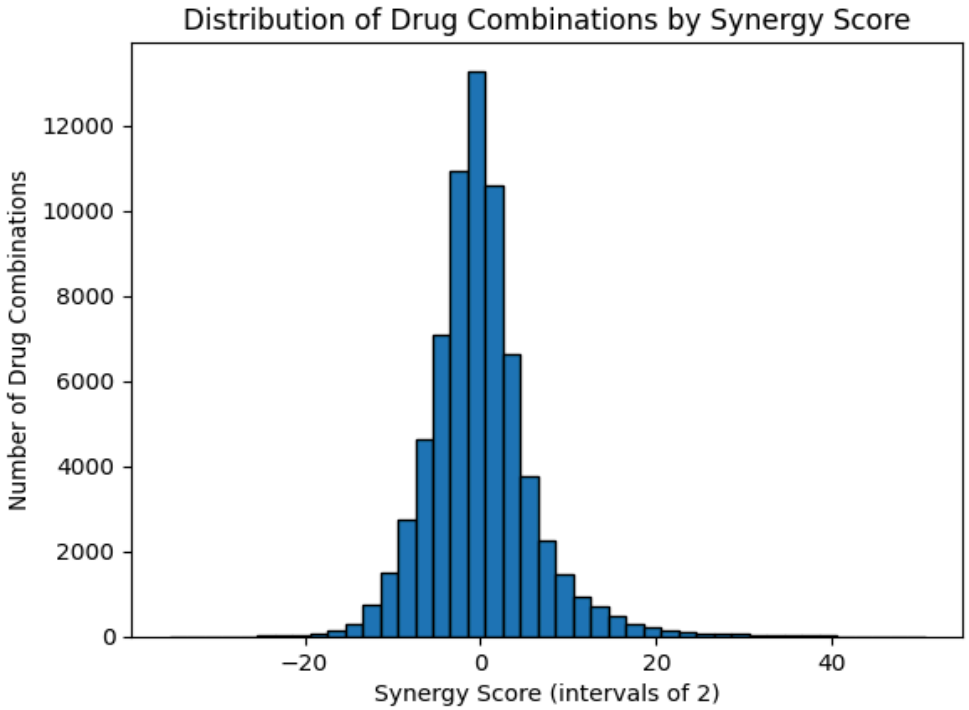
Over all drugs and cells in total the distribution of labels was reasonably balanced, however you can see that some drugs have a pretty strong label imbalance.

Distribution of Synergy Scores

We were given a large dataset, one of them containing two drugs and a synergy score when they are combined. Looking at the distribution of synergy scores we got the following graph:

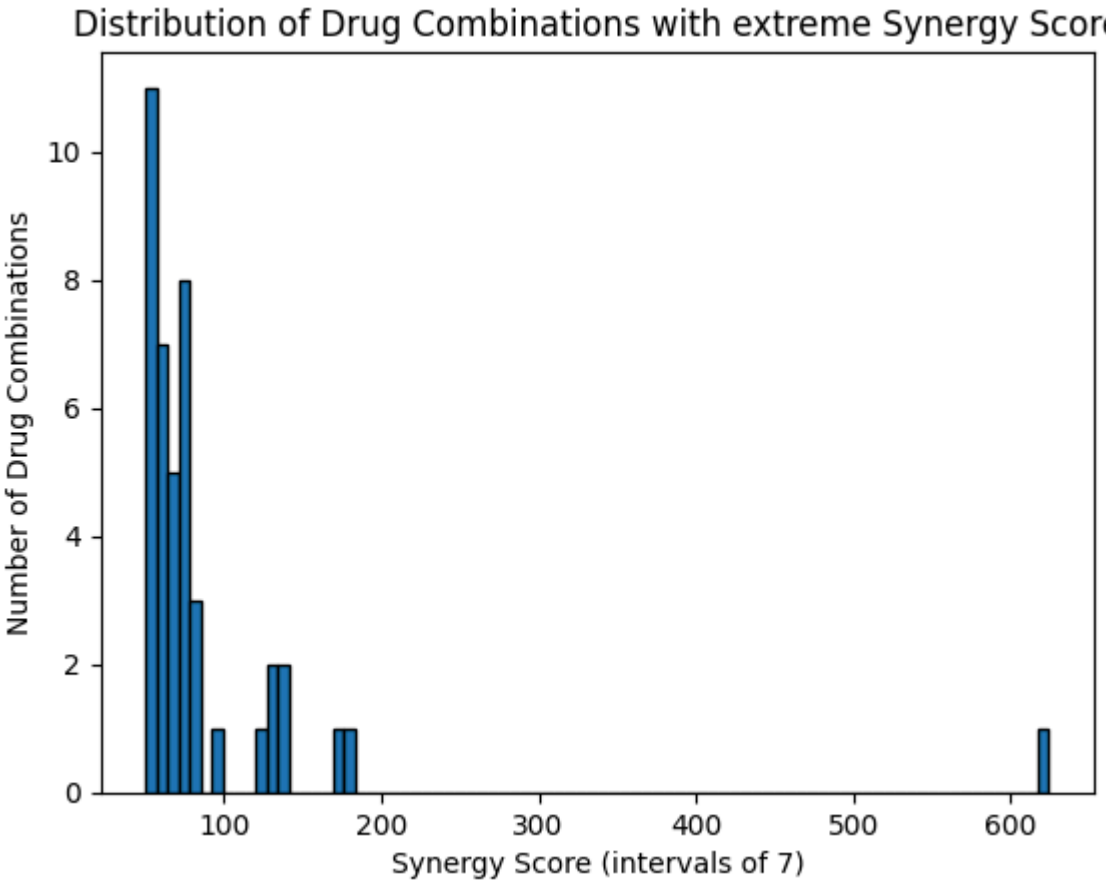


The graph is heavily skewed by the fact that some scores are extremely high. When looking only at the scores below 50, we get the following graph:



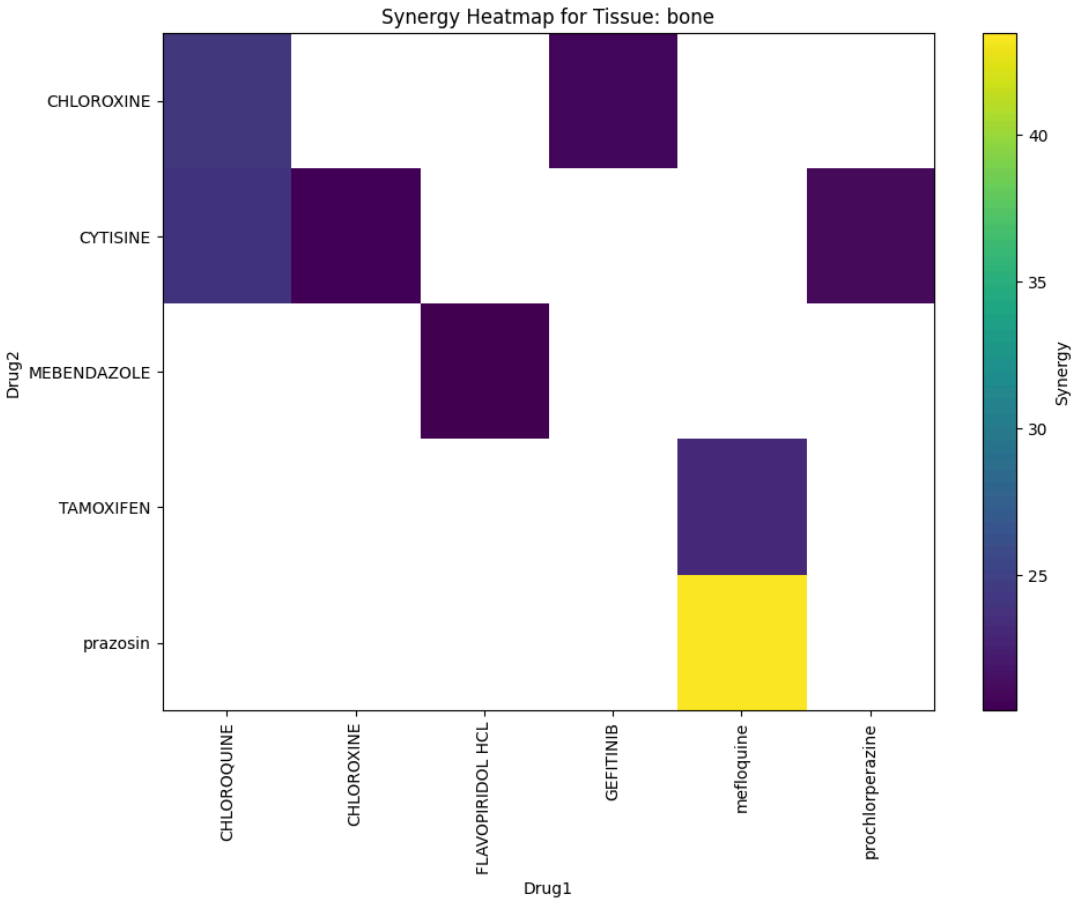
This graph shows that the scores are normally distributed, which is a good sign as it is what we would expect given a correct dataset.

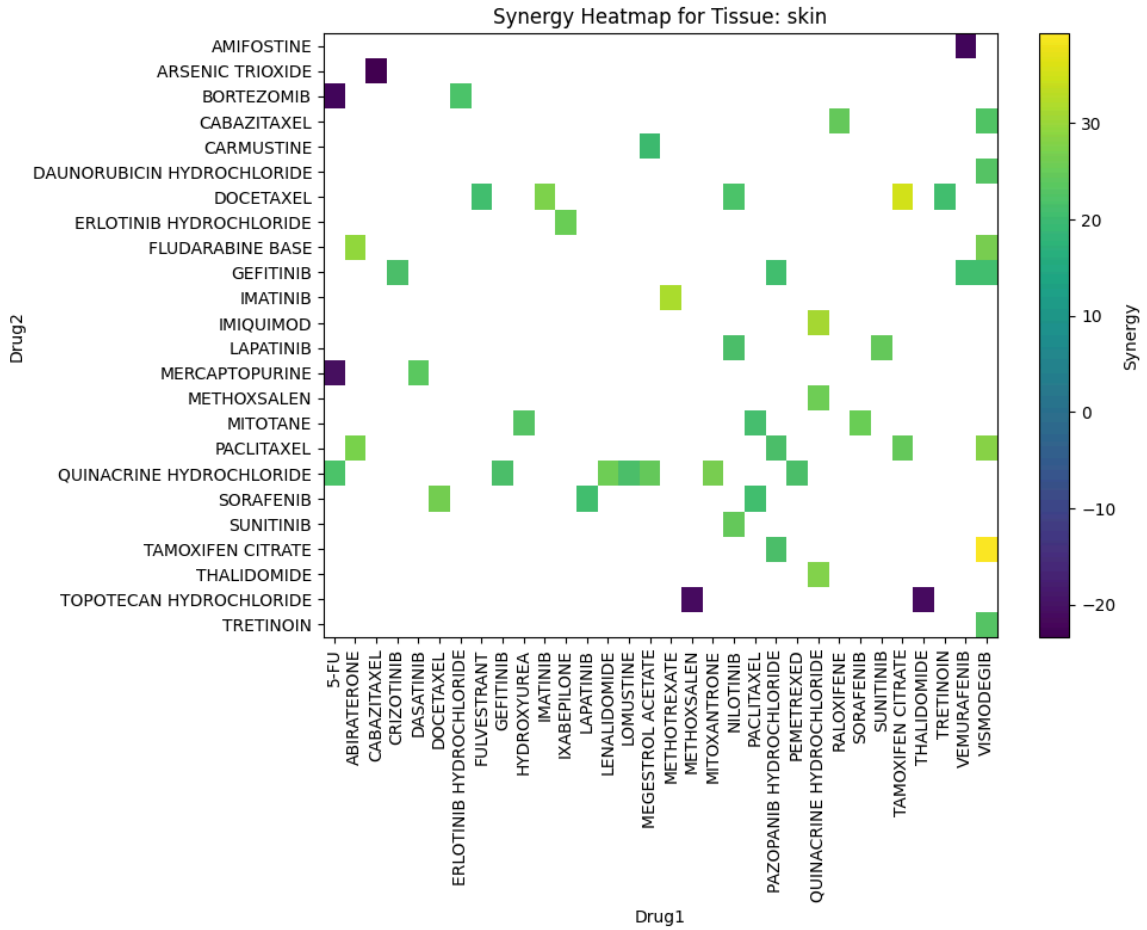
Looking at the outliers above a score of 50 however seem to indicate errors as the data looks random.

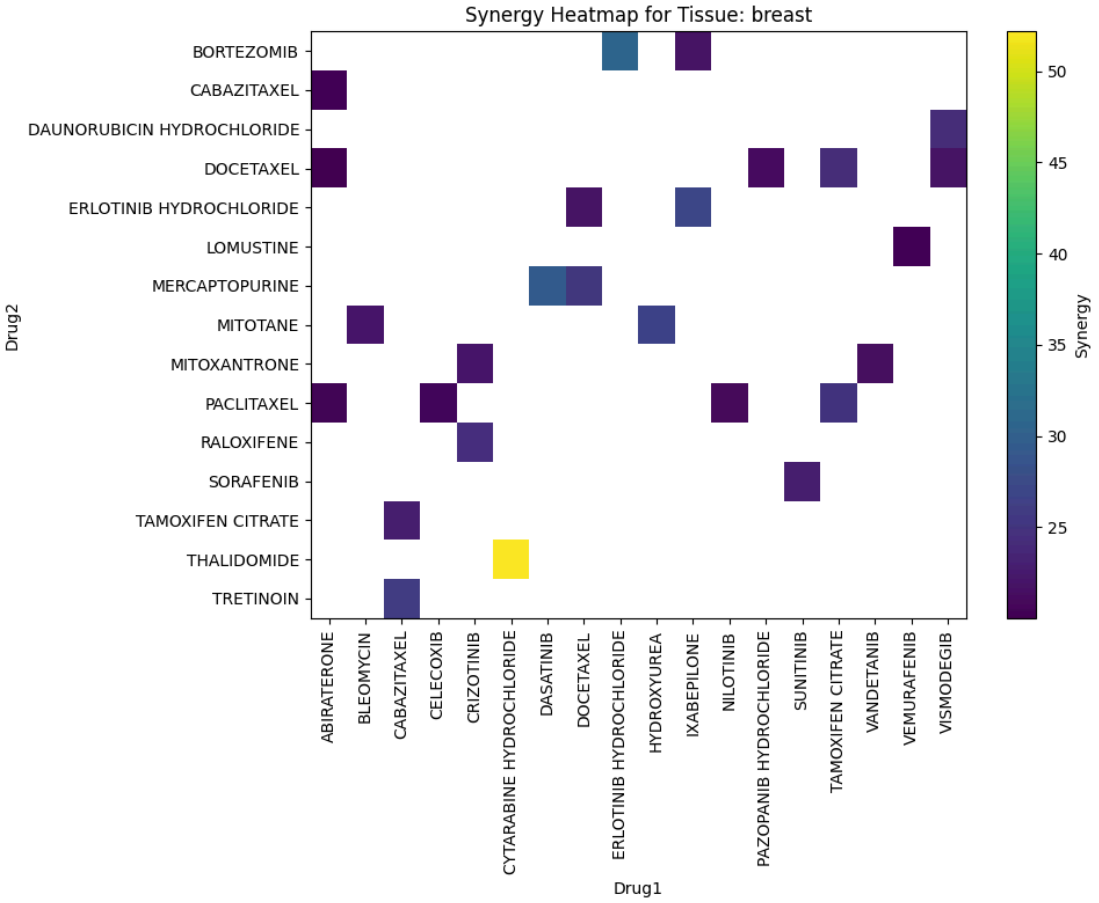


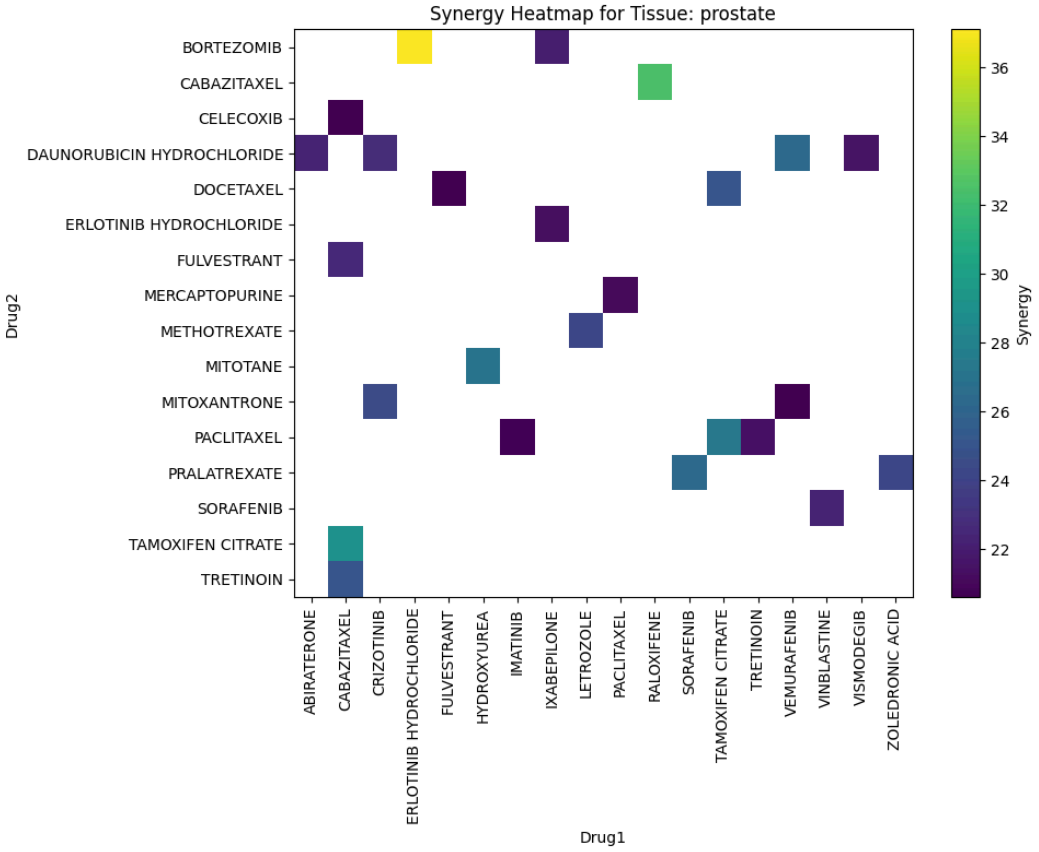
However these outliers are not many, with the highest bar reaching only 11 occurrences, and the datasets contain almost 70 000 data points. This indicates that the outliers may simply be extreme cases without being errors.

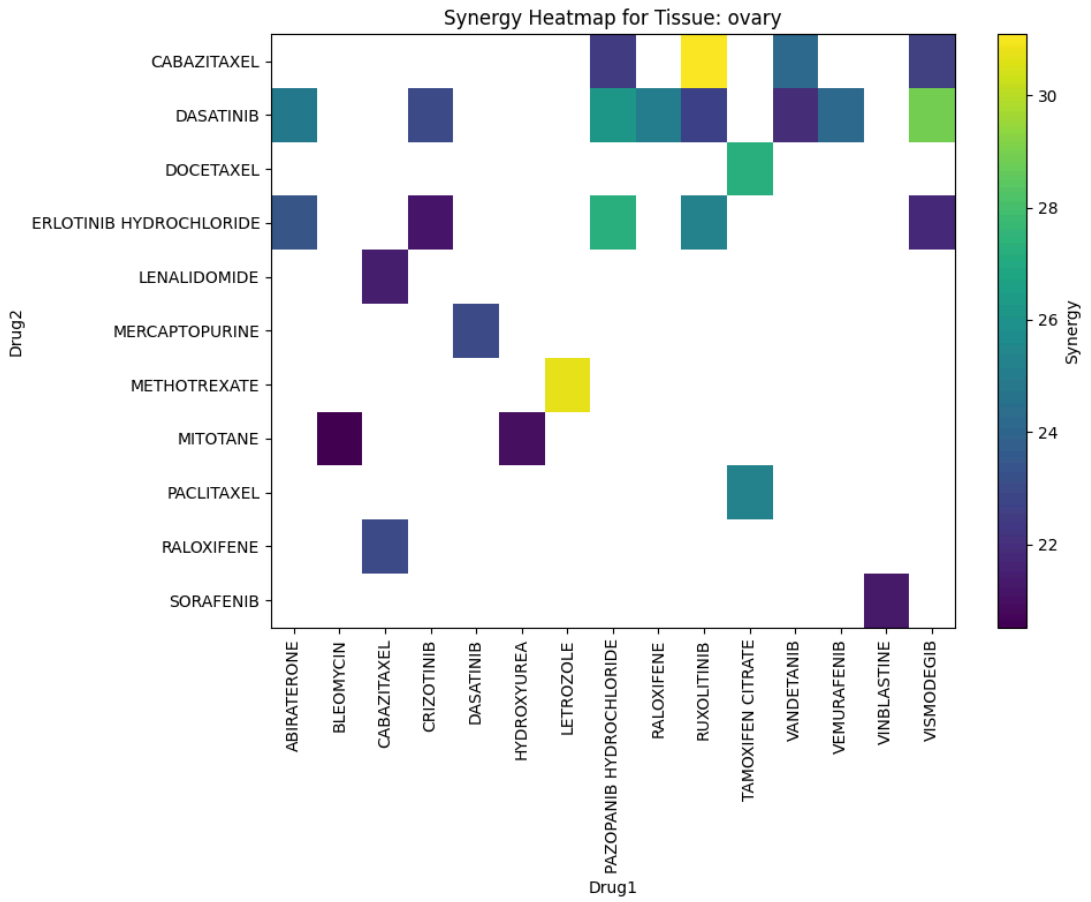
Looking closer at synergy scores from each tissue, we get the following heatmaps:

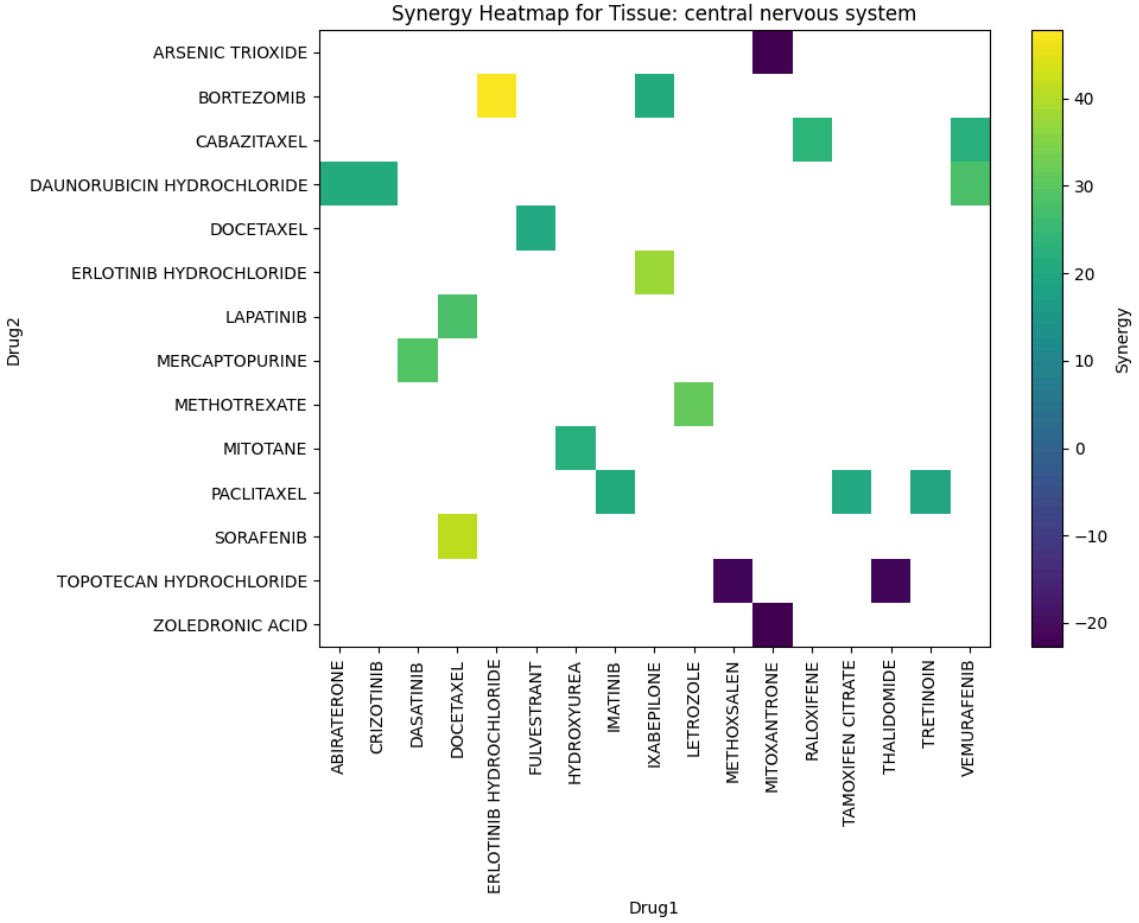


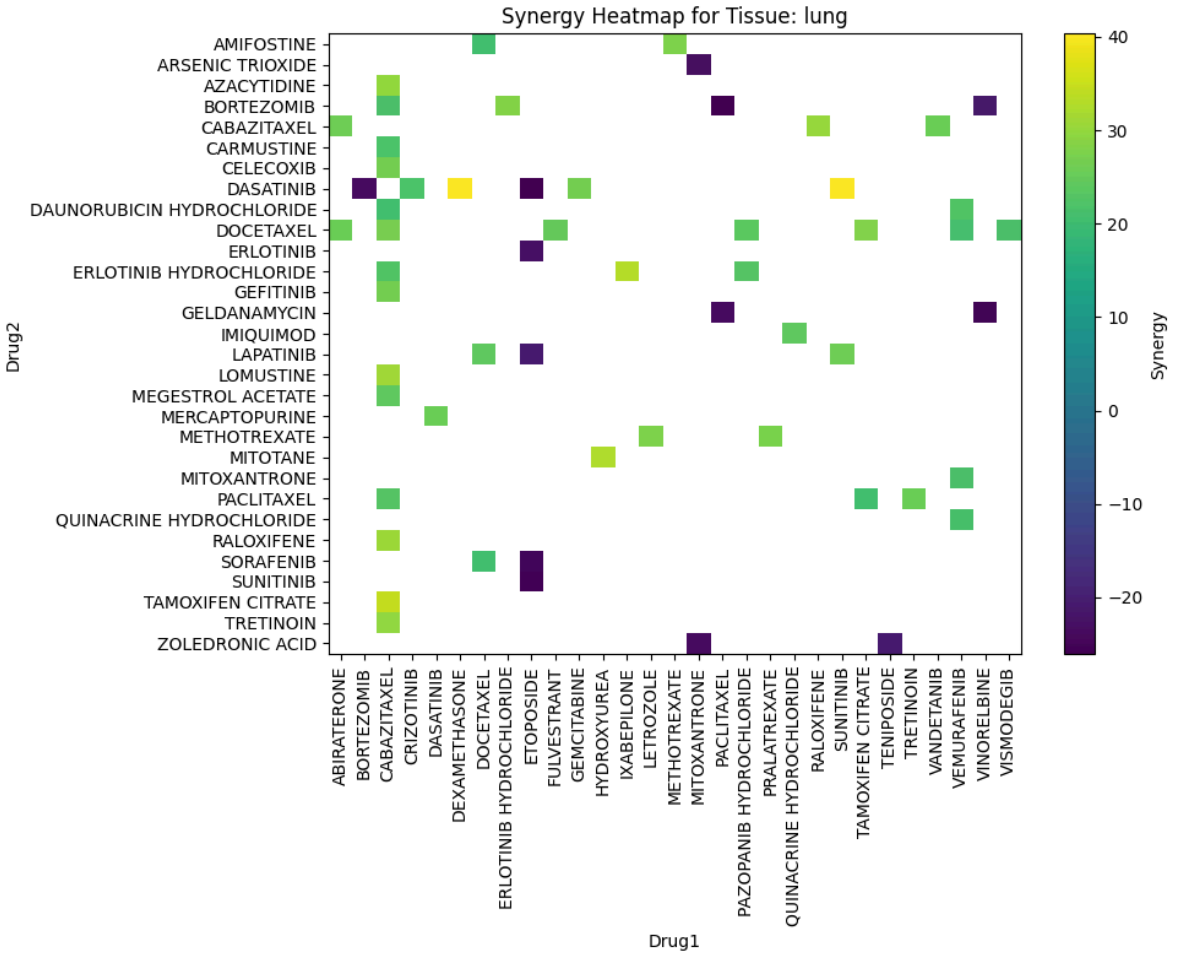


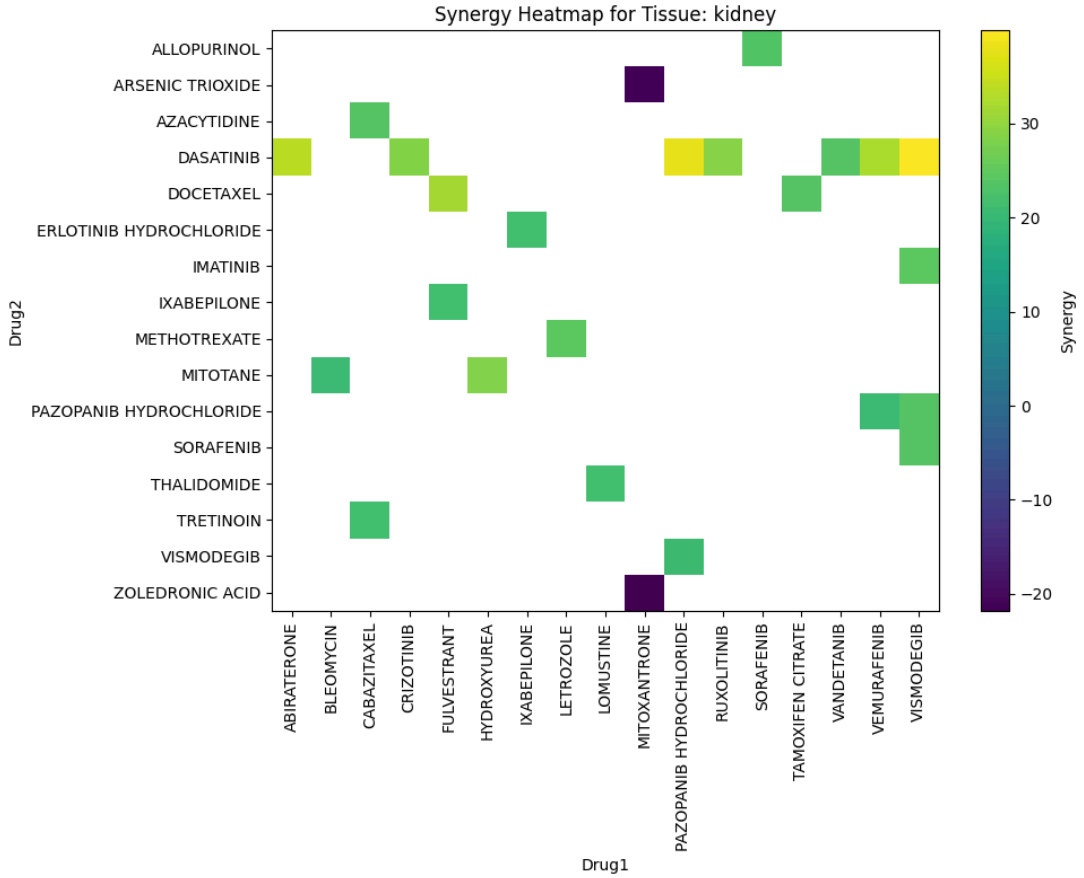

Synergy Heatmap for Tissue: breast
BORTEZOMIB
CABAZITAXEL
DAUNORUBICIN HYDROCHLORIDE
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
LOMUSTINE
MERCAPTOPURINE
MITOTANE
MITOXANTRONE
PACLITAXEL
RALOXIFENE
SORAFENIB
TAMOXIFEN CITRATE
THALIDOMIDE
TRETINOIN
Drug2
ABIRATERONE
BLEOMYCIN
CABAZITAXEL
CELECOXIB
CRIZOTINIB
CYTARABINE HYDROCHLORIDE
DASATINIB
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
HYDROXYUREA
IXABEPILONE
NILOTINIB
PAZOPANIB HYDROCHLORIDE
SUNITINIB
TAMOXIFEN CITRATE
VANDETANIB
VEMURAFENIB
VISMODEGIB
Drug1
Synergy
50
45
40
35
30
25

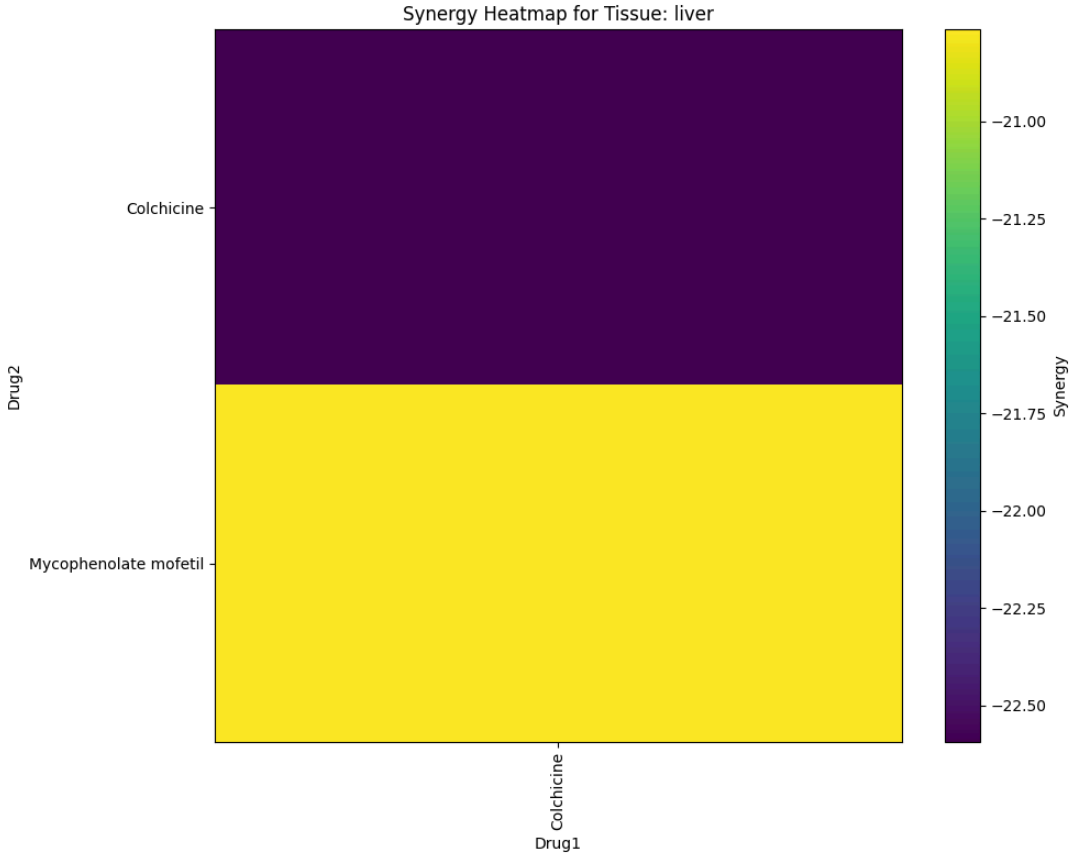

Synergy Heatmap for Tissue: prostate
BORTEZOMIB
CABAZITAXEL
CELECOXIB
DAUNORUBICIN HYDROCHLORIDE
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
FULVESTRANT
MERCAPTOPURINE
METHOTREXATE
MITOTANE
MITOXANTRONE
PACLITAXEL
PRALATREXATE
SORAFENIB
TAMOXIFEN CITRATE
TRETINOIN
Drug2
ABIRATERONE
CABAZITAXEL
CRIZOTINIB
ERLOTINIB HYDROCHLORIDE
FULVESTRANT
HYDROXYUREA
IMATINIB
IXABEPILONE
LETROZOLE
PACLITAXEL
RALOXIFENE
SORAFENIB
TAMOXIFEN CITRATE
TRETINOIN
VEMURAFENIB
VINBLASTINE
VISMODEGIB
ZOLEDRONIC ACID
Drug1
Synergy
36
34
32
30
28
26
24
22

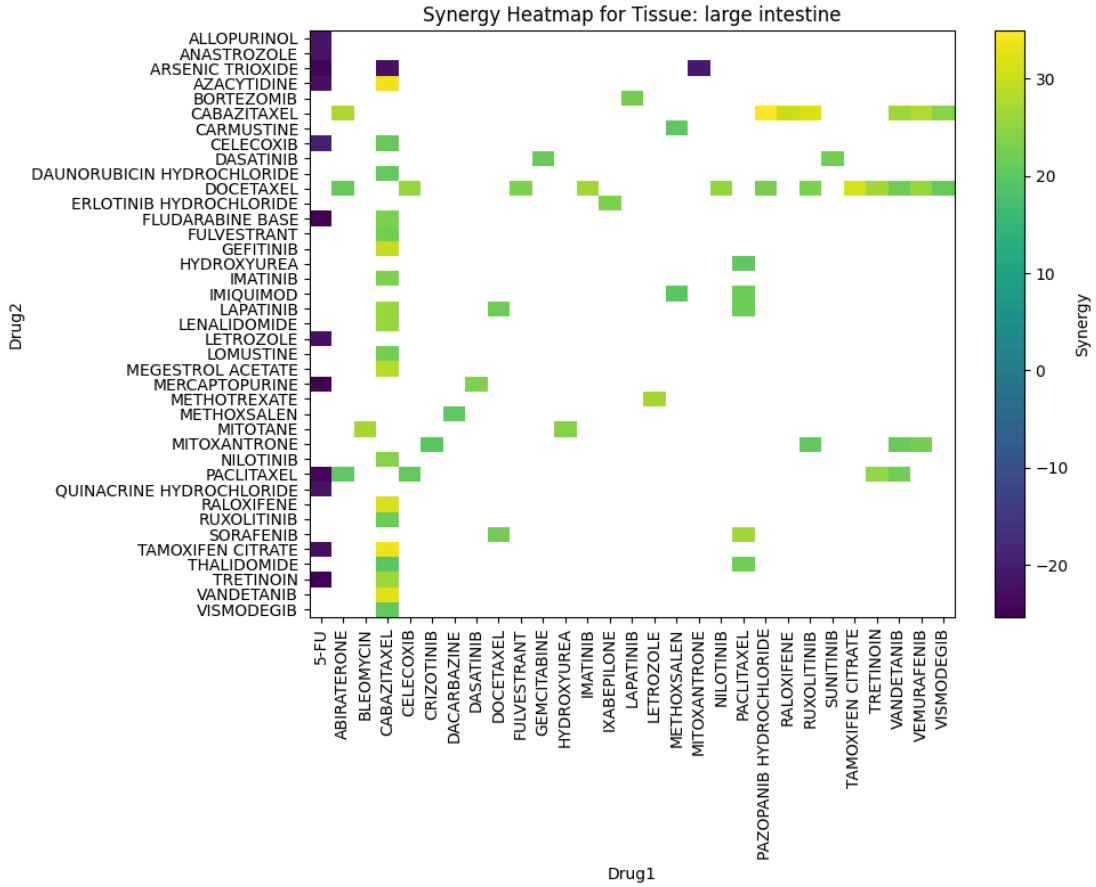

Synergy Heatmap for Tissue: ovary
CABAZITAXEL
DASATINIB
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
LENALIDOMIDE
MERCAPTOPURINE
METHOTREXATE
MITOTANE
PACLITAXEL
RALOXIFENE
SORAFENIB
Drug2
ABIRATERONE
BLEOMYCIN
CABAZITAXEL
CRIZOTINIB
DASATINIB
HYDROXYUREA
LETROZOLE
PAZOPANIB HYDROCHLORIDE
RALOXIFENE
RUXOLITINIB
TAMOXIFEN CITRATE
VANDETANIB
VEMURAFENIB
VINBLASTINE
VISMODEGIB
Drug1
30
28
26
24
22
Synergy

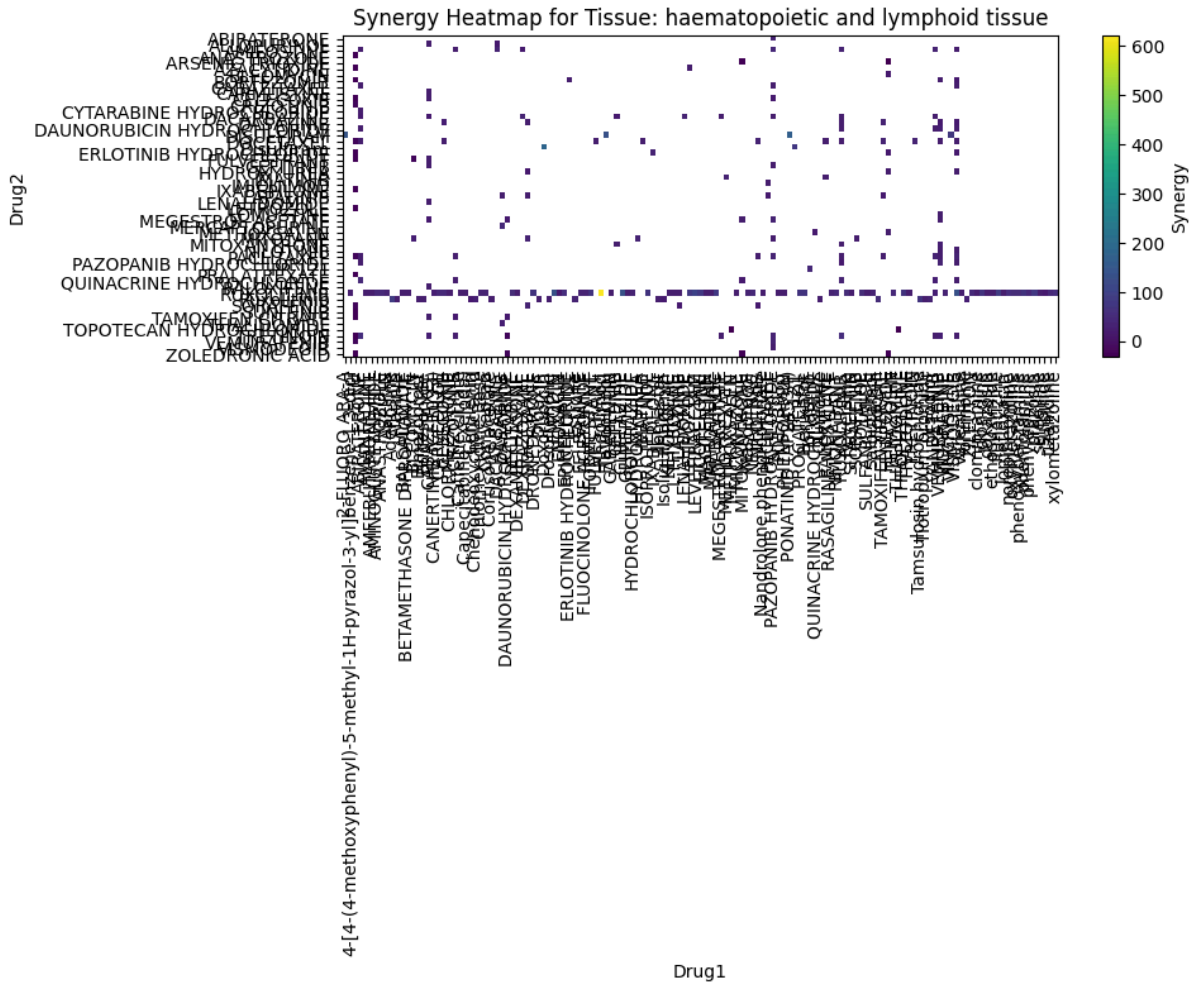

Synergy Heatmap for Tissue: central nervous system
ARSENIC TRIOXIDE
BORTEZOMIB
CABAZITAXEL
DAUNORUBICIN HYDROCHLORIDE
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
LAPATINIB
MERCAPTOPURINE
METHOTREXATE
MITOTANE
PACLITAXEL
SORAFENIB
TOPOTECAN HYDROCHLORIDE
ZOLEDRONIC ACID
Drug2
ABIRATERONE
CRIZOTINIB
DASATINIB
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
FULVESTRANT
HYDROXYUREA
IMATINIB
IXABEPILONE
LETROZOLE
METHOXSALEN
MITOXANTRONE
RALOXIFENE
TAMOXIFEN CITRATE
THALIDOMIDE
TRETINOIN
VEMURAFENIB
Drug1
40
30
20
10
0
−10
−20
Synergy


Synergy Heatmap for Tissue: lung
Drug2
AMIFOSTINE
ARSENIC TRIOXIDE
AZACYTIDINE
BORTEZOMIB
CABAZITAXEL
CARMUSTINE
CELECOXIB
DASATINIB
DAUNORUBICIN HYDROCHLORIDE
DOCETAXEL
ERLOTINIB
ERLOTINIB HYDROCHLORIDE
GEFITINIB
GELDANAMYCIN
IMIQUIMOD
LAPATINIB
LOMUSTINE
MEGESTROL ACETATE
MERCAPTOPURINE
METHOTREXATE
MITOTANE
MITOXANTRONE
PACLITAXEL
QUINACRINE HYDROCHLORIDE
RALOXIFENE
SORAFENIB
SUNITINIB
TAMOXIFEN CITRATE
TRETINOIN
ZOLEDRONIC ACID
ABIRATERONE
BORTEZOMIB
CABAZITAXEL
CRIZOTINIB
DASATINIB
DEXAMETHASONE
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
ETOPOSIDE
FULVESTRANT
GEMCITABINE
HYDROXYUREA
IXABEPILONE
LETROZOLE
METHOTREXATE
MITOXANTRONE
PACLITAXEL
PAZOPANIB HYDROCHLORIDE
PRALATREXATE
QUINACRINE HYDROCHLORIDE
RALOXIFENE
SUNITINIB
TAMOXIFEN CITRATE
TENIPOSIDE
TRETINOIN
VANDETANIB
VEMURAFENIB
VINORELBINE
VISMODEGIB
Drug1
Synergy
40
30
20
10
0
−10
−20


Synergy Heatmap for Tissue: kidney
Drug2
ALLOPURINOL
ARSENIC TRIOXIDE
AZACYTIDINE
DASATINIB
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
IMATINIB
IXABEPILONE
METHOTREXATE
MITOTANE
PAZOPANIB HYDROCHLORIDE
SORAFENIB
THALIDOMIDE
TRETINOIN
VISMODEGIB
ZOLEDRONIC ACID
ABIRATERONE
BLEOMYCIN
CABAZITAXEL
CRIZOTINIB
FULVESTRANT
HYDROXYUREA
IXABEPILONE
LETROZOLE
LOMUSTINE
MITOXANTRONE
PAZOPANIB HYDROCHLORIDE
RUXOLITINIB
SORAFENIB
TAMOXIFEN CITRATE
VANDETANIB
VEMURAFENIB
VISMODEGIB
Drug1
Synergy
30
20
10
0
−10
−20

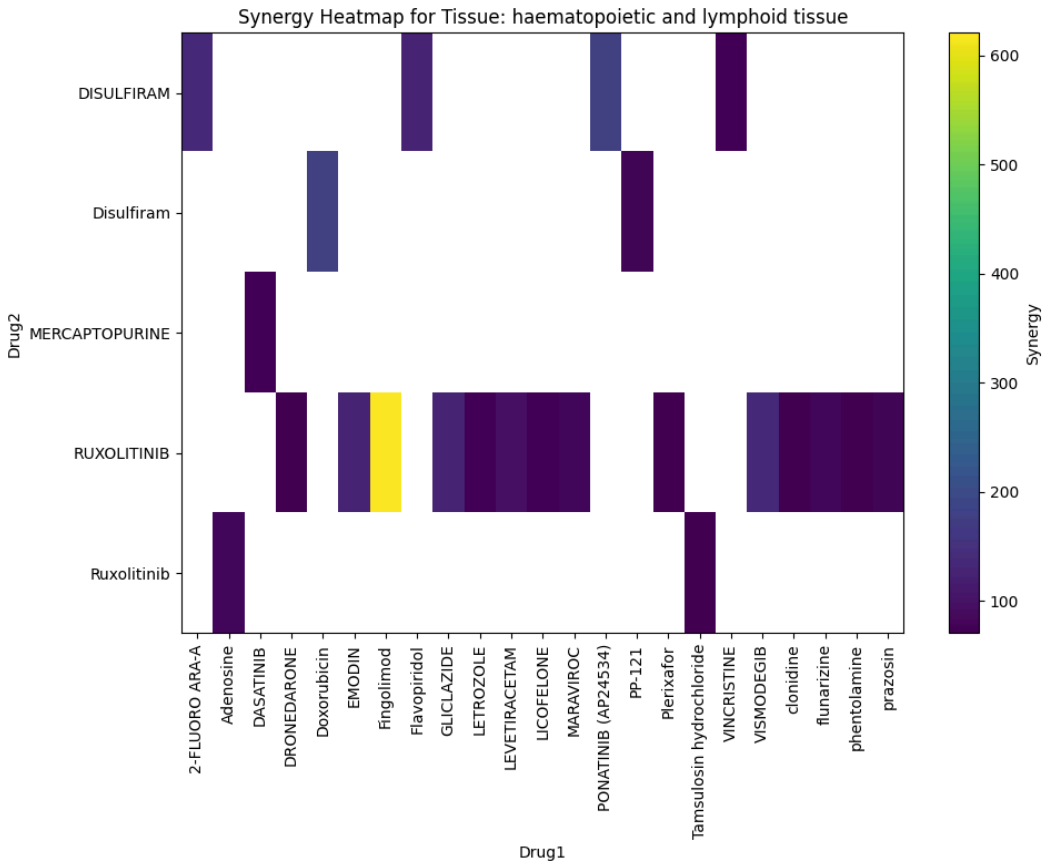

Synergy Heatmap for Tissue: liver
Colchicine
Mycophenolate mofetil
Drug2
Colchicine
Drug1
−21.00
−21.25
−21.50
−21.75
−22.00
−22.25
−22.50
Synergy


Synergy Heatmap for Tissue: large intestine
ALLOPURINOL
ANASTROZOLE
ARSENIC TRIOXIDE
AZACYTIDINE
BORTEZOMIB
CABAZITAXEL
CARMUSTINE
CELECOXIB
DASATINIB
DAUNORUBICIN HYDROCHLORIDE
DOCETAXEL
ERLOTINIB HYDROCHLORIDE
FLUDARABINE BASE
FULVESTRANT
GEFITINIB
HYDROXYUREA
IMATINIB
IMIQUIMOD
LAPATINIB
LENALIDOMIDE
LETROZOLE
LOMUSTINE
MEGESTROL ACETATE
MERCAPTOPURINE
METHOTREXATE
METHOXSALEN
MITOTANE
MITOXANTRONE
NILOTINIB
PACLITAXEL
QUINACRINE HYDROCHLORIDE
RALOXIFENE
RUXOLITINIB
SORAFENIB
TAMOXIFEN CITRATE
THALIDOMIDE
TRETINOIN
VANDETANIB
VISMODEGIB
Drug2
5-FU
ABIRATERONE
BLEOMYCIN
CABAZITAXEL
CELECOXIB
CRIZOTINIB
DACARBAZINE
DASATINIB
DOCETAXEL
FULVESTRANT
GEMCITABINE
HYDROXYUREA
IMATINIB
IXABEPILONE
LAPATINIB
LETROZOLE
METHOXSALEN
MITOXANTRONE
NILOTINIB
PACLITAXEL
PAZOPANIB HYDROCHLORIDE
RALOXIFENE
RUXOLITINIB
SUNITINIB
TAMOXIFEN CITRATE
TRETINOIN
VANDETANIB
VEMURAFENIB
VISMODEGIB
Drug1
30
20
10
0
−10
−20
Synergy



Findings

Most of the heatmaps have synergy scores ranging from roughly -20 to 50. Hematopoietic and lymphoid tissue however have scores reaching all the way to 600, and have too large of a dataset to make sense of the heatmap. Let's look closer at its extreme scores:



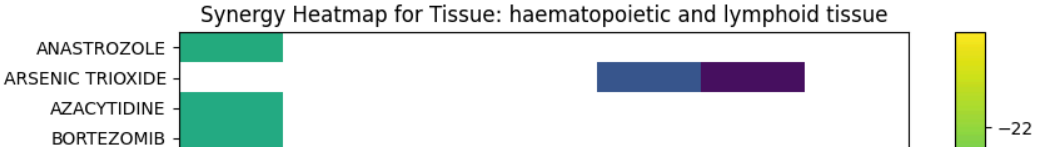
Scores above 70:

Scores below -20:

Hematopoietic and lymphoid tissue clearly has some outliers, there are several data points that falls outside the range of the other tissues

Outlier Removal

Considering the occurrence of outliers, we want to see if they are errors that pollute the data and negatively impact the models performance. To remove the outliers, we used the interquartile range (IQR) method. The IQR method helps find and remove outliers by focusing on the middle 50% of the data. It first calculates the first quartile (Q1) and third quartile (Q3), which are the values at 25% and 75% of the dataset. The interquartile range is the difference between them (IQR = Q3 - Q1). Normally, any value below Q1 - 1.5 × IQR or above Q3 + 1.5 × IQR is considered an outlier. However, different multipliers can be used depending on how strict you want the outlier detection to be. A smaller multiplier (e.g., 0.5) catches fewer outliers, while a



larger multiplier (e.g., 3.0) allows more variation before flagging a value as an outlier. To apply the IQR method to the model, we simply amend the load_data function to filter the data:

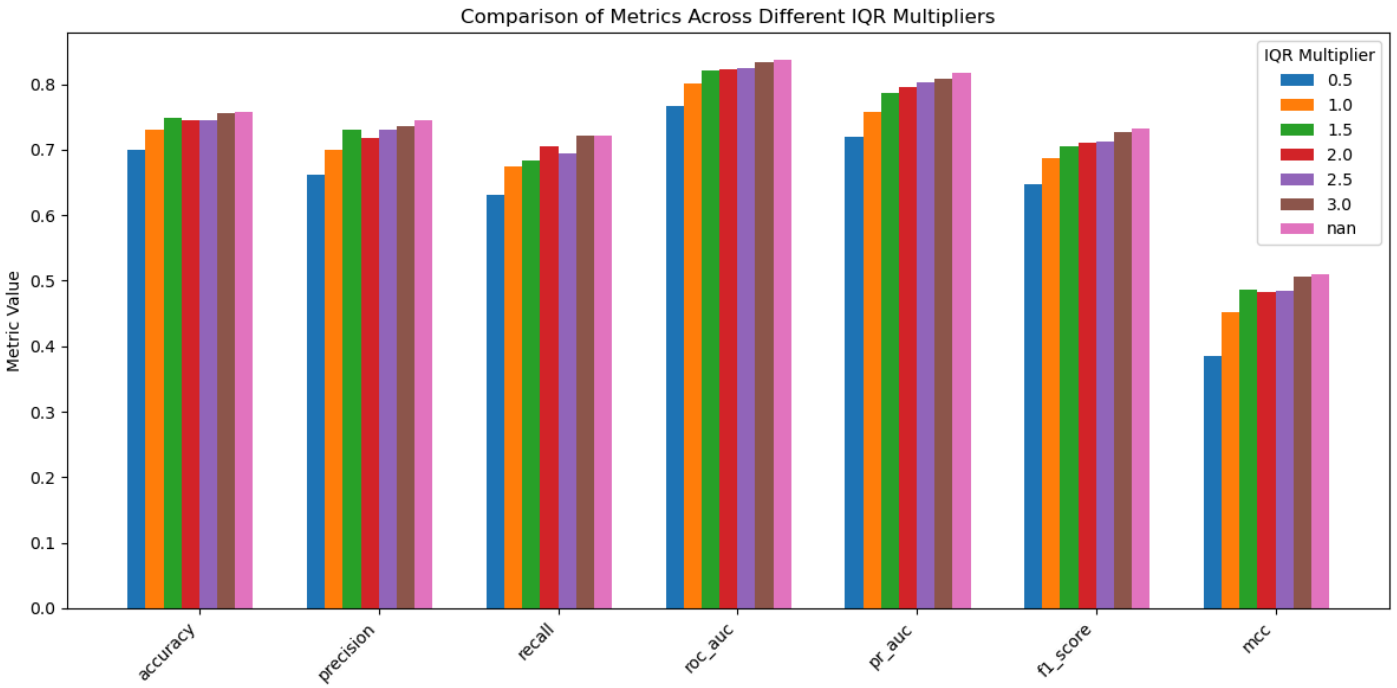
```python
def remove_outliers(self, df, column, iqr_multiplyer):
    """Removes outliers from a DataFrame based on the IQR method."""
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - iqr_multiplyer * IQR
    upper_bound = Q3 + iqr_multiplyer * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

def load_data(self):
    drug_combination_df = pd.read_csv(os.path.join(self.data_dir, 'drug_combinations.csv'))
    ppi_df = pd.read_excel(os.path.join(self.data_dir, 'protein-protein_network.xlsx'))
    cpi_df = pd.read_csv(os.path.join(self.data_dir, 'cell_protein.csv'))
    dpi_df = pd.read_csv(os.path.join(self.data_dir, 'drug_protein.csv'))

    drug_combination_df = self.remove_outliers(drug_combination_df, 'synergy', 3.0)

    return drug_combination_df, ppi_df, cpi_df, dpi_df
```

We tested different IQR multipliers ranging from 0.5 to 3.0 with intervals of 0.5. Running the model with the different IQR multipliers, we got the following results:



A higher the multiplier, means that the method is less strict of what it considers an outlier. The “nan” multiplier means simply that no outliers are removed. The test results show that the model performs better when no outliers are removed. This indicates that the outliers are most likely not in fact errors, but just extreme cases.

This confirms the null-hypothesis – that removing outliers will not affect the model's performance positively – and therefore, proceeding from here, we will not remove outliers when running the model.

Cross-validation with Leakage Free Folds

In order to correctly estimate the prediction performance of any neural network and to mitigate the risks associated with overfitting, it is important to employ a suitable cross-validation scheme. Cross-validation is a well established validation strategy that ensures the model has the ability to generalize to data unseen and unrelated to the training dataset by systematically partitioning the dataset into different subsets i. e. "*folds*" and iteratively training, validating and testing the model on each of these folds.

In our approach, we employed k-folds cross validation, splitting the dataset into 5 folds. At each iteration we used 3 folds as training data, 1 fold as validation data, to adjust hyperparameter configurations during training, and 1 fold as testing data, on which predictions were performed with the trained model. After testing is done at the end of each iteration step, the roles of the folds are shifted and the process is repeated. The process repeats 5 times, until each fold has been used as a test set exactly once. The final evaluation metrics presented in the subsequent graphs, refer to the mean and standard deviations of predictions across the test folds.

Preventing Data Leakage

A key aspect to consider during cross validation is to ensure the created folds are leakage free, meaning information in one fold is disjoint from information contained in any other fold. If this is not addressed, the model would simply predict what it already learned, i. e. memorize interactions from the training data rather than learning meaningful patterns, which would lead to an artificially inflated evaluation of the models true generalization capabilities.

In our case, a trade-off was necessary between a fair evaluation, maintaining data diversity and implementation complexity:

- A fully disjoint set of folds would add tremendous technical implementation complexity and lead to a large number of folds to ensure complete disjointness, given the number of interacting components present within the dataset. This would result in a reduction in data diversity across folds.

- Not implementing a relatively rigorous fold separation, would lead to the aforementioned overestimation of the model performance

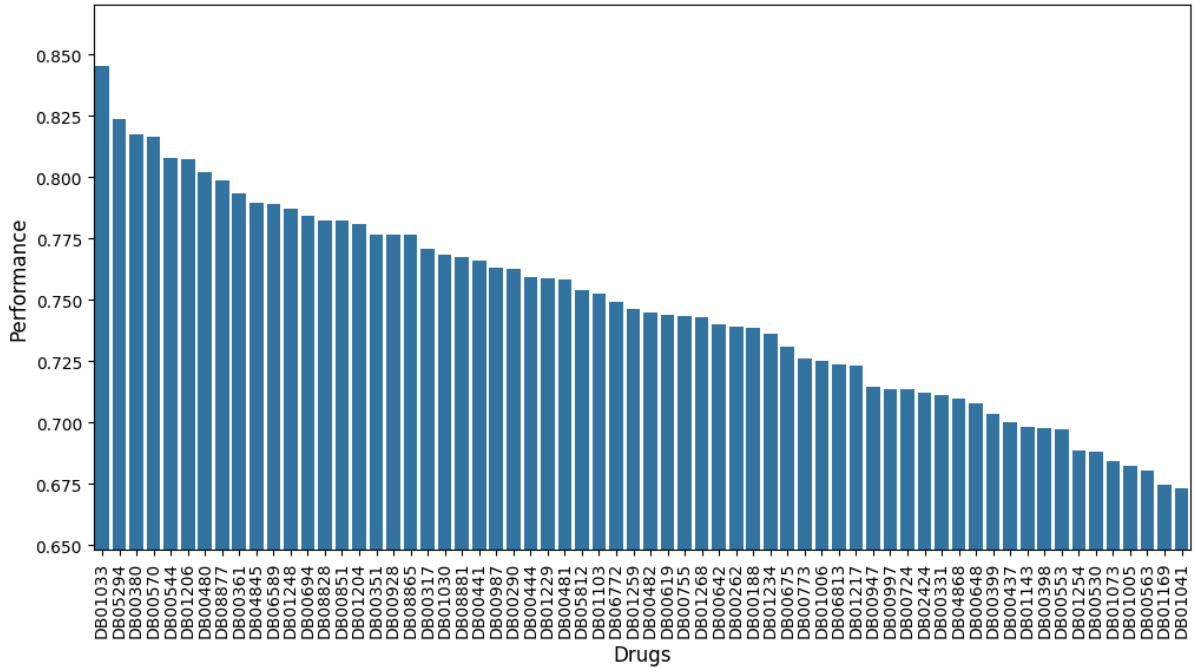
In our approach, we separated the folds based on unique drug combinations, where all instances containing a combination of (DrugA, DrugB), would be located in only one fold. Multiple unique drug combinations are allowed in the same fold, however if a combination appears in a specific fold, it can not appear in any other fold. This approach does not ensure completely disjoint folds, information about a drug can leak if that drug appears in more than one combination. However, for our proposed ablation studies on the model, this approach provides sufficient insights into the model's learning capabilities while still allowing for meaningful comparisons across different configurations.

Model Performance analysis

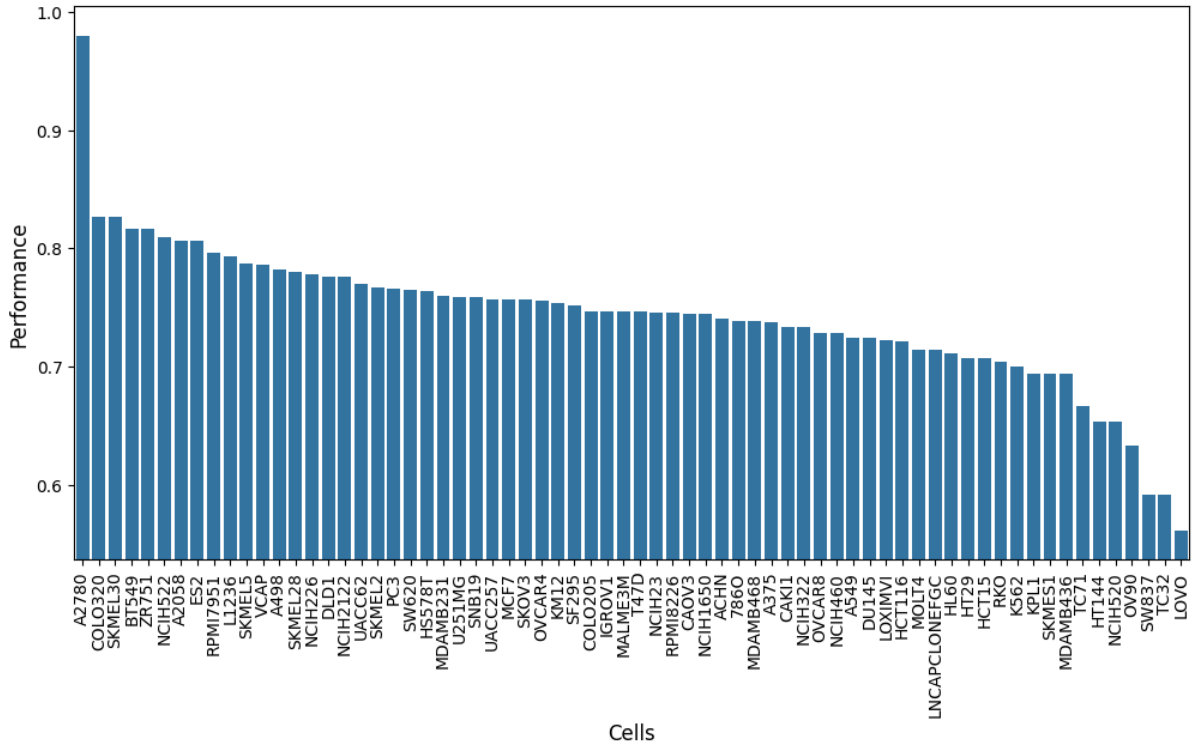
	validation loss	accuracy	precision	recall
Default	0.52	0.76	0.74	0.73
Simple	0.71	0.73	0.71	0.71
Graphless	0.52	0.75	0.73	0.72
CV Default	0.54	0.75	0.73	0.71
CV Simple	0.84	0.7	0.67	0.68
CV Graphless	0.54	0.75	0.73	0.71

Model Performance comparison (CV = Cross Validation)

After implementing cross-validation, we compared the results across different models. We also included a simple baseline model, which is described in the Alternative Models section. One of these models, called "Graphless," does not use the graph structure that the original model relies on. Essentially, it removes the additional precision that graph information could provide. Surprisingly, excluding the graph had little to no impact on the model's performance. This suggests that the paper's claim that the graph improves model accuracy may not hold true in practice.



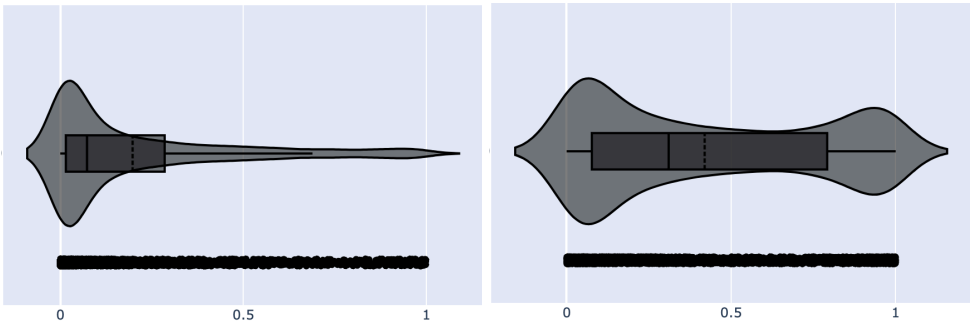
Drug accuracy Cross Validation Default model DrugCombDB



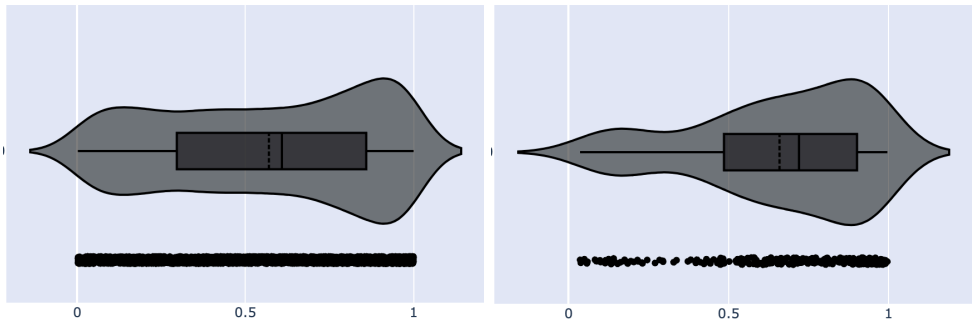
Cell accuracy cross validation default model DrugCombDB

Next, we wanted to investigate whether a large portion of the model's accuracy could be attributed to label imbalance. To explore this, we analyzed the model's performance on individual cells and drugs. We plotted the highest-performing cells and drugs and compared them to the average performance to see if any patterns

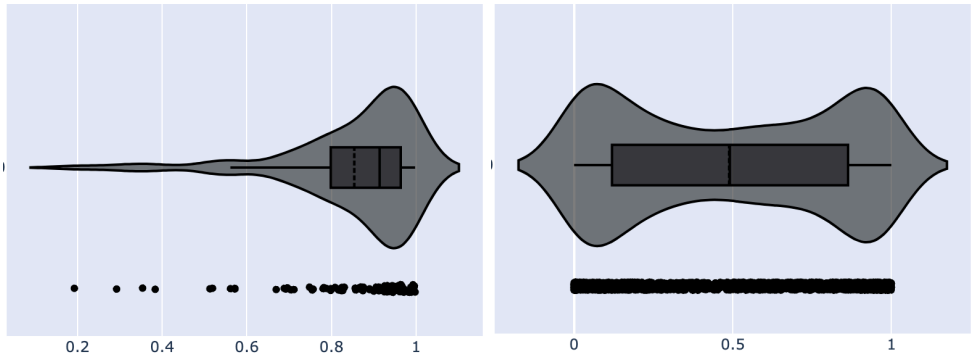
emerged from the analysis.



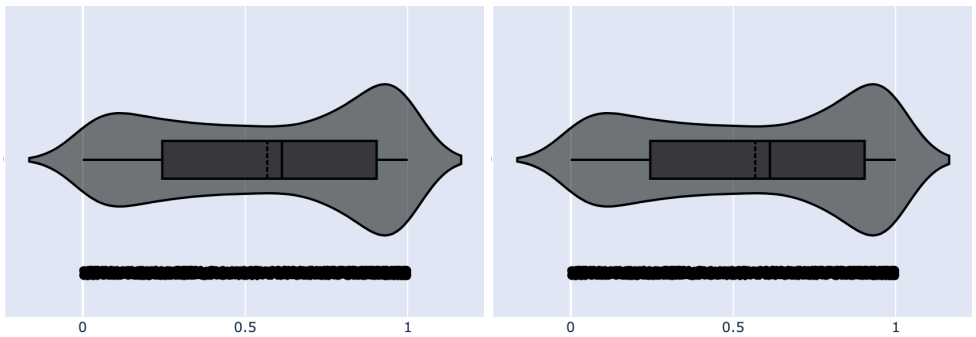
High accuracy drug prediction distribution



High accuracy drug prediction distribution



High accuracy cell prediction distribution



Average accuracy cell prediction distribution

The conclusion was that there is no clear or recurring pattern. However, we observed that for cases with higher accuracy, the prediction distributions were sharper—showing more predictions close to 0 or 1 and fewer between 0.4 and 0.6. This indicates that the model achieves most of its accuracy from factors beyond simple statistical biases like label imbalance.

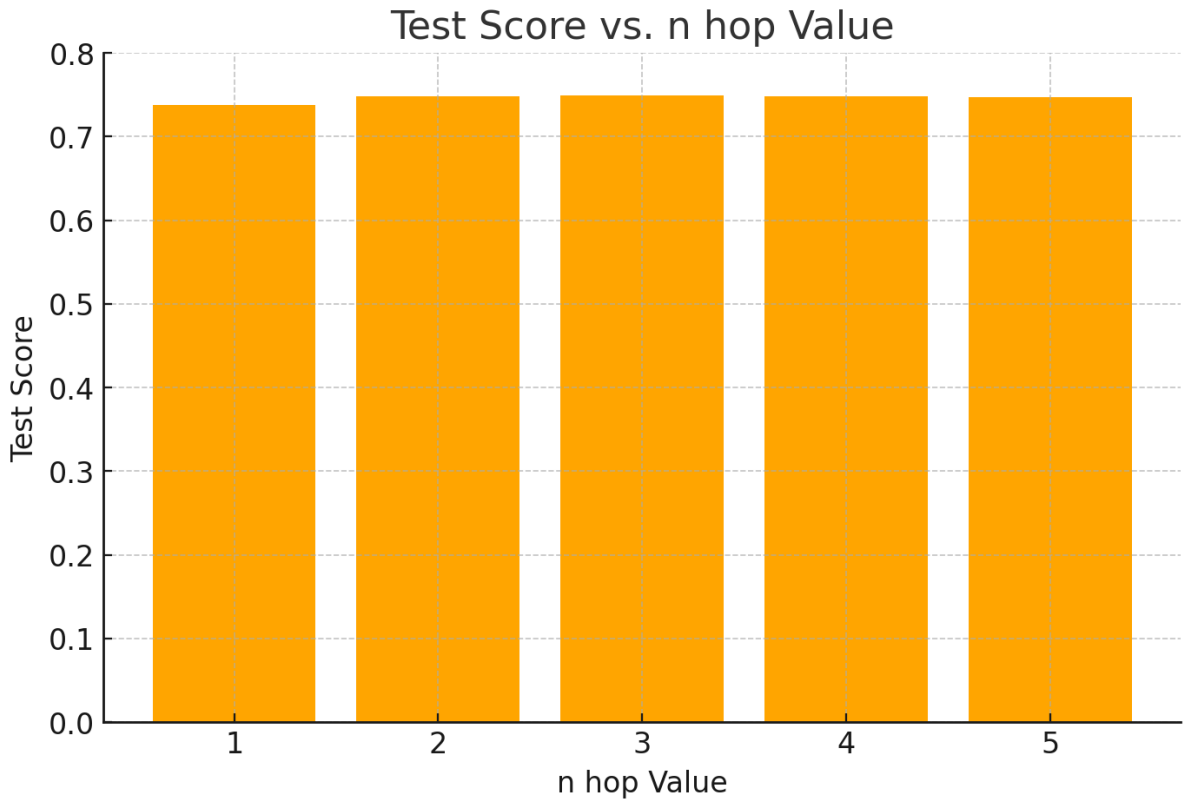
Hyperparameter Tuning

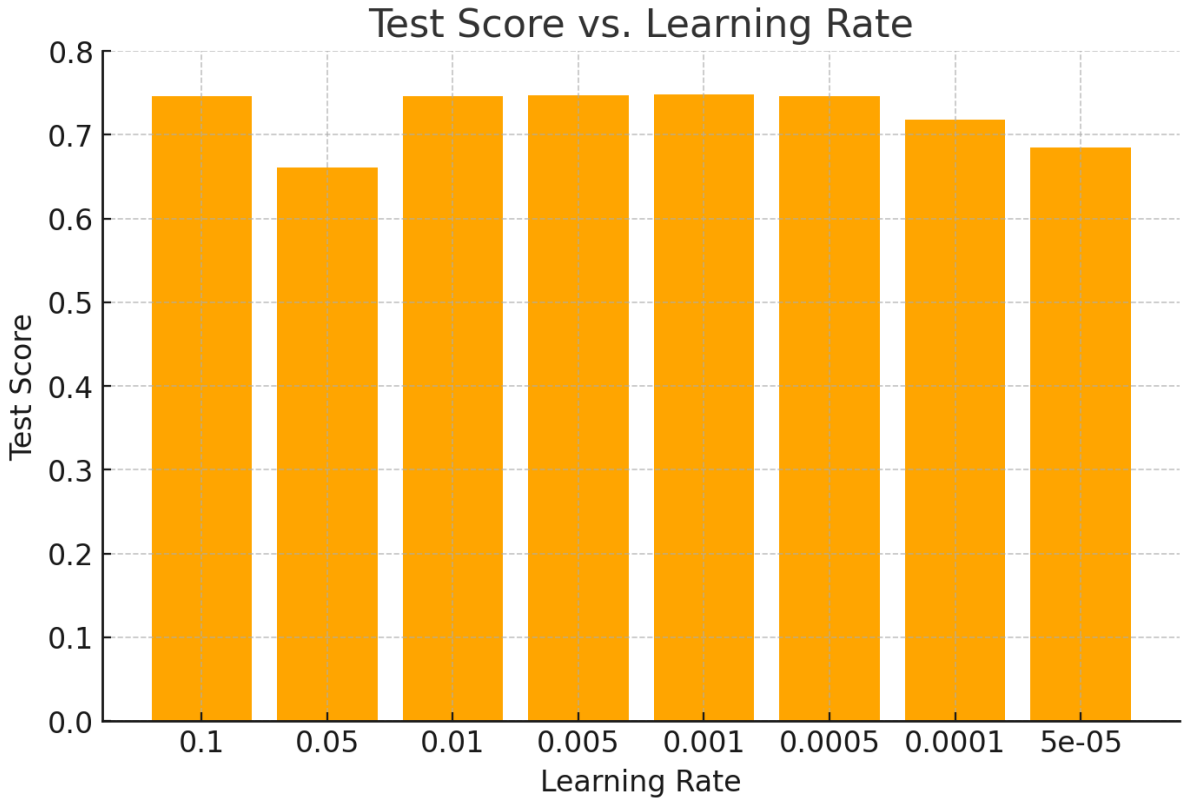
The hyper parameters for this models are as following:

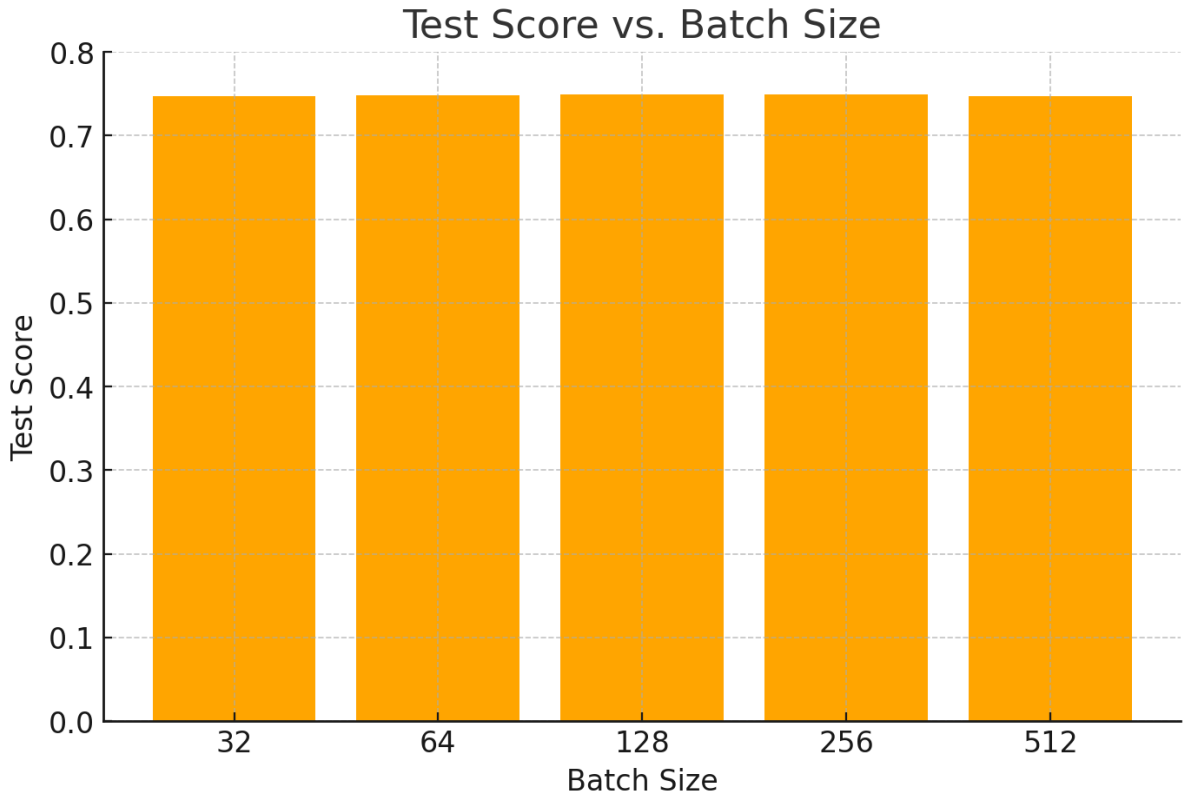
- **n hop**: Depth of graph neighborhood aggregation.
- **Learning rate:** Controls the step size during weight updates.
- **Batch size:** Number of samples processed together in one iteration.
- **Embedding Dimension:** Number of samples processed together in one iteration.
- **Weight Decay:** Regularization strength to penalize large weights.
- **Number of Memory Units:** Size of memory used for graph processing.
- **Therapy strategy:** Determines the strategy used to combine drug and cell embeddings for synergy prediction.
- **L1 decay:** regularization that adds a penalty to the model for having large weights, encouraging the model to make some weights exactly zero, which can simplify the model and reduce overfitting.

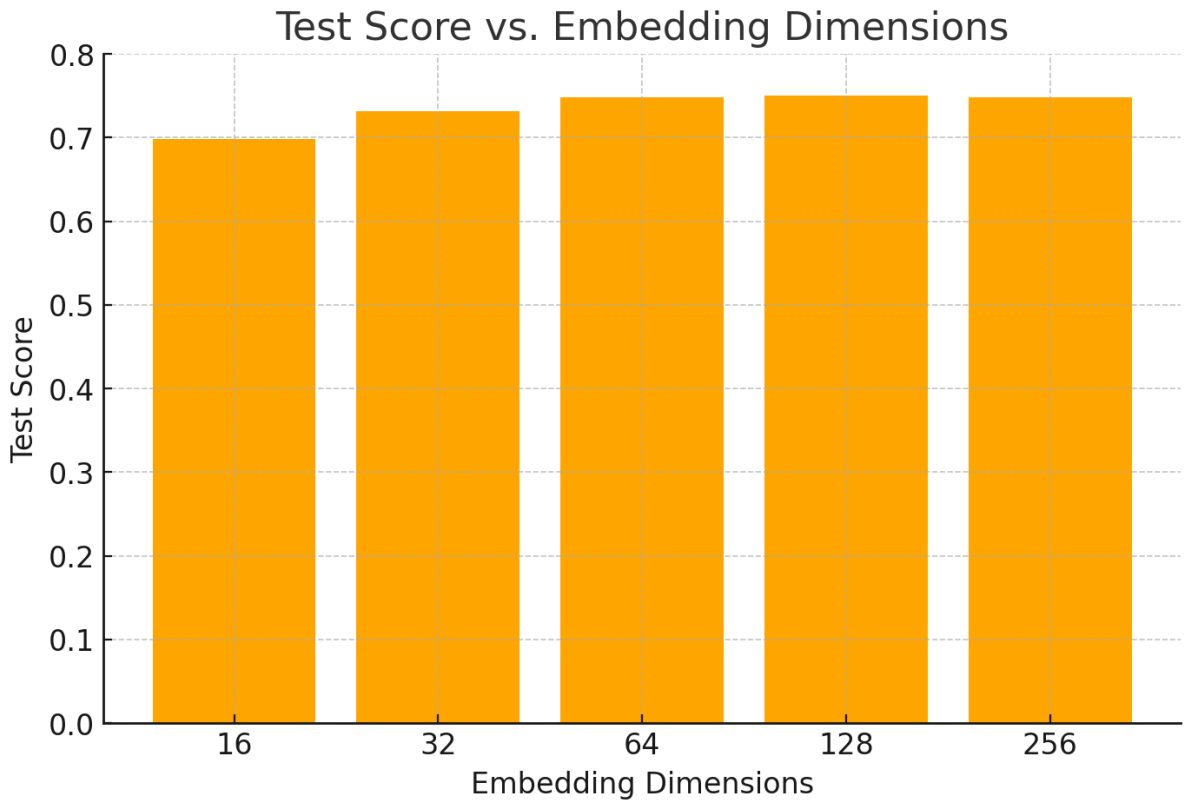
Tuning Parameters Individually

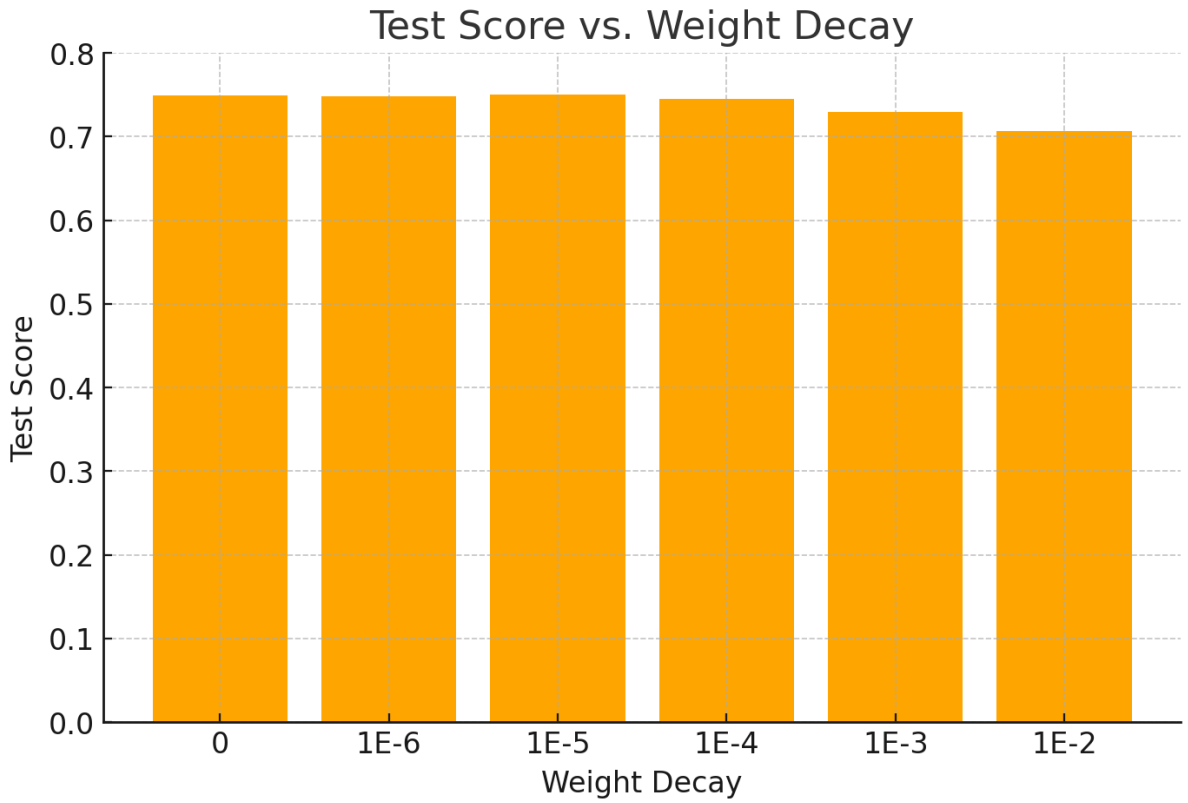
We tested different values for each parameter and ran the model to optimize all. Here are the results:

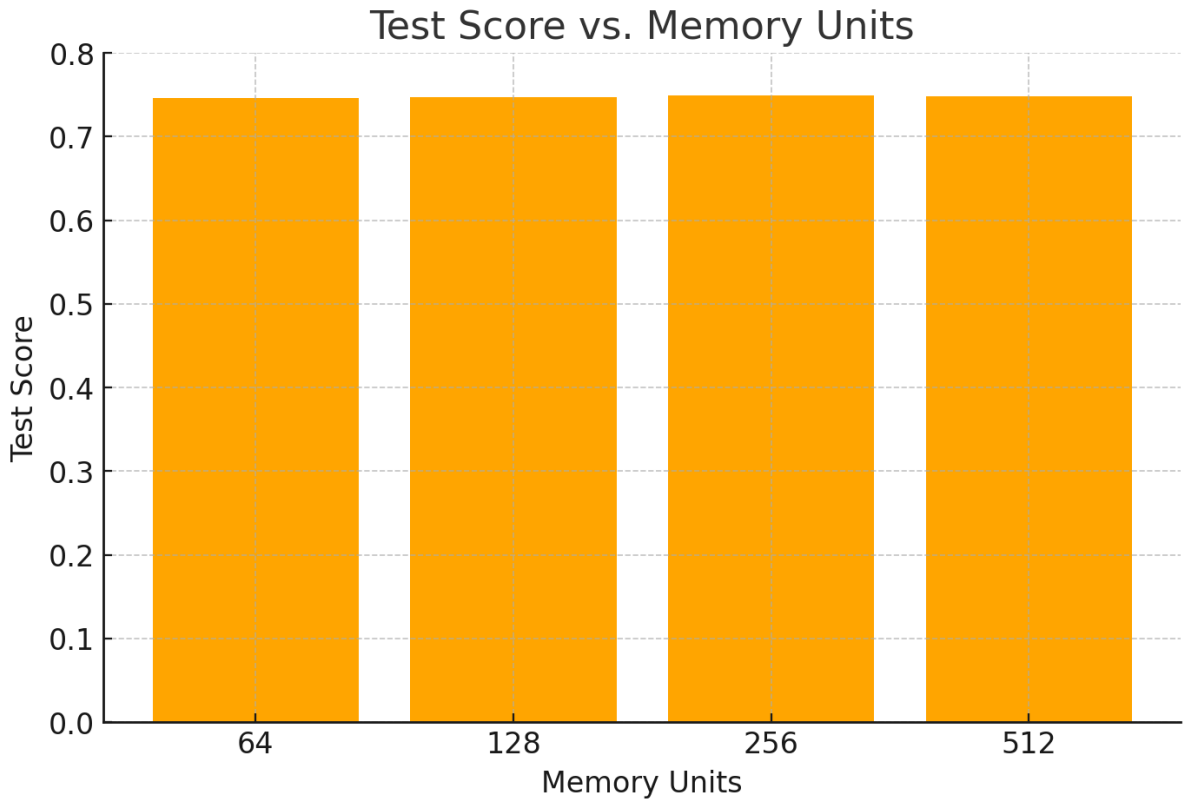


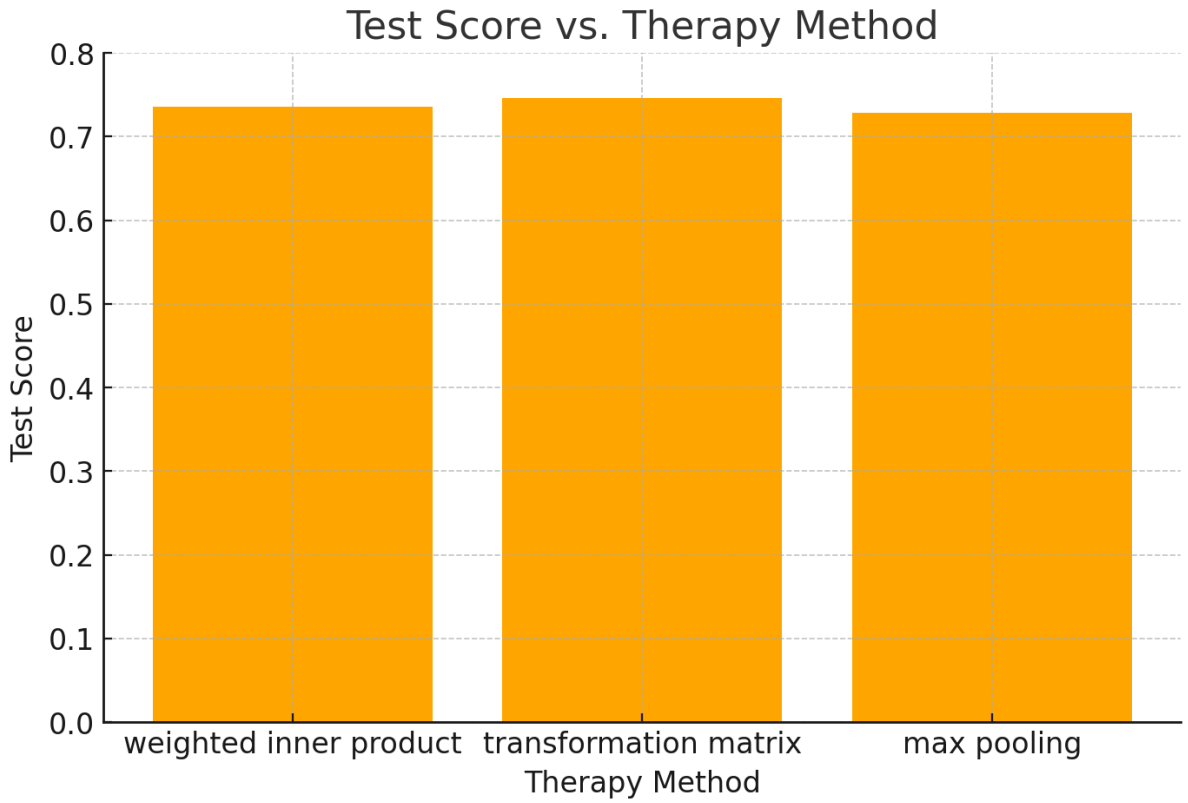


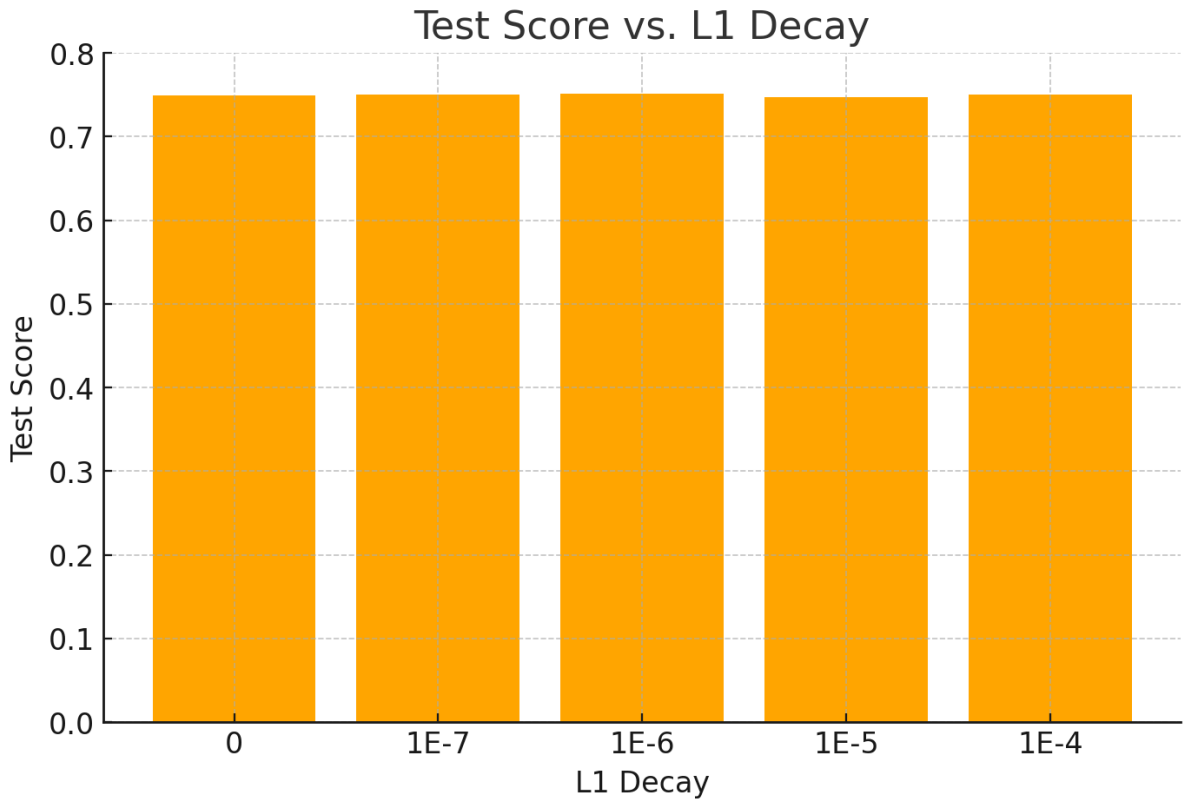

Test Score vs. Batch Size
Test Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
32
64
128
256
512
Batch Size


Test Score vs. Embedding Dimensions
Test Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
16
32
64
128
256
Embedding Dimensions


Test Score vs. Weight Decay
Test Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1E-6
1E-5
1E-4
1E-3
1E-2
Weight Decay


Test Score vs. Memory Units
Test Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
64
128
256
512
Memory Units


Test Score vs. Therapy Method
Test Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
weighted inner product
transformation matrix
max pooling
Therapy Method


Test Score vs. L1 Decay
Test Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1E-7
1E-6
1E-5
1E-4
L1 Decay

As the charts show, changing the values for most parameters does not affect the model's performance drastically. There are some differences however. Then we used all the best scores from each parameter and got the following results:

Improvement									
	Training			Validation			Test		
Values	Acc	Prec	Score	Acc	Prec	Score	Accuracy	Precision	Score
Average of all originals	0,82125	0,8125	0,816875	0,76	0,74125	0,750625	0,7537265068	0,7418026977	0,7477646022
Optimized	0,83	0,82	0,825	0,76	0,74	0,75	0,7556707712	0,7442741666	0,7499724689
Improvement	0,00875	0,0075	0,008125	0	−0,00125	−0,000625	0,0019442644	0,0024714689	0,0022078667
% improvement	1,07%	0,92%	0,74	0,00%	−0,17%	0,74	0,26%	0,33%	0,30%

Test score increased by 0.3% compared to the average of the base model. This is a very miniscule improvement, and it did fall within the range of variance of scores we have gathered running the base model throughout the project, so the improvement is not statistically significant.

Hyperparameter Sweeps

In addition to tuning hyperparameters individually with all else remaining as is to evaluate their effects on performance, it is also important to perform hyper parameter tuning in unison, testing multiple hyperparameter configurations simultaneously. This is important, because it is often the case that changing one hyperparameter might affect the behaviour of another hyperparameter (e. g. if we change the activation function, we might also have to optimize the dropout ratio accordingly).

This process can be cumbersome and time consuming, especially when dealing with large neural networks dependent on various hyperparameters. To automate this process, we took advantage of the Weights & Biases API (wandb), which provides an automation platform for ML training and inference workloads. Leveraging wandb's Python library, we employed what are referred to as "*hyperparameter sweeps*"–an automated method of iterating through different combinations of hyperparameter values to compare the results of different combinations and find an optimal performing configuration. This provides a structured approach to finding the optimal set of hyperparameters and understanding interactions between hyperparameters while minimizing manual trial and error.

Sweep Configuration and Optimization Strategy

```
"sweep_config": {
    "name": "DrugCombDB Sweep 3 (Essentials)",
    "description": "Optimize emb_dim, n_hop, n_memory, for ROC-AUC",
    "method": "bayes",
    "metric": {
        "name": "val_roc_auc",
        "goal": "maximize"
    },
    "parameters": {
        "emb_dim": {
            "values": [64, 128, 256]
        },
        "n_hop": {
            "values": [2, 3, 4]
        },
        "n_memory": {
            "values": [64, 128, 256]
        }
    },
    "early_terminate": {
        "type": "hyperband",
        "min_iter": 1,
        "eta": 3
    }
}
```

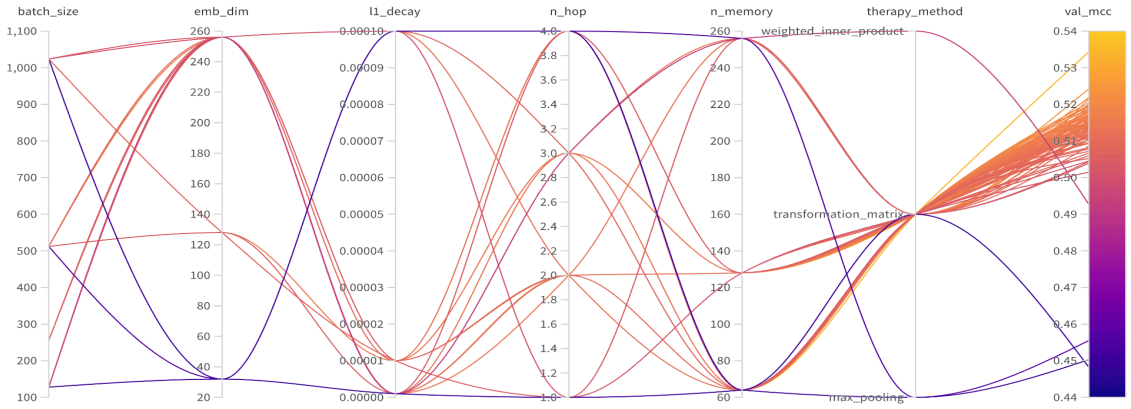
The Hyperparameter Sweeps were configured using a config file similar to the one in the given figure. Using this configuration scheme, an automated and efficient search strategy was implemented to explore a broad range of values. The config setup consists of the following:

- **Name and Description of the Sweep**
- **Value Ranges For The Hyperparameters:**
    - Consists of the names of the parameters and a search space for each parameter. The search space can either be a list of predefined values or a range of discrete/continuous values depending on the parameter type.
- **The Search Strategy (Method):**
    - Bayesian optimization was used to intelligently navigate the search space
    - Rather than combining parameter values randomly and trying every possible combination, this approach starts out with randomly selected values, builds a probabilistic model of the performance, and refines the search towards more promising hyperparameter values.

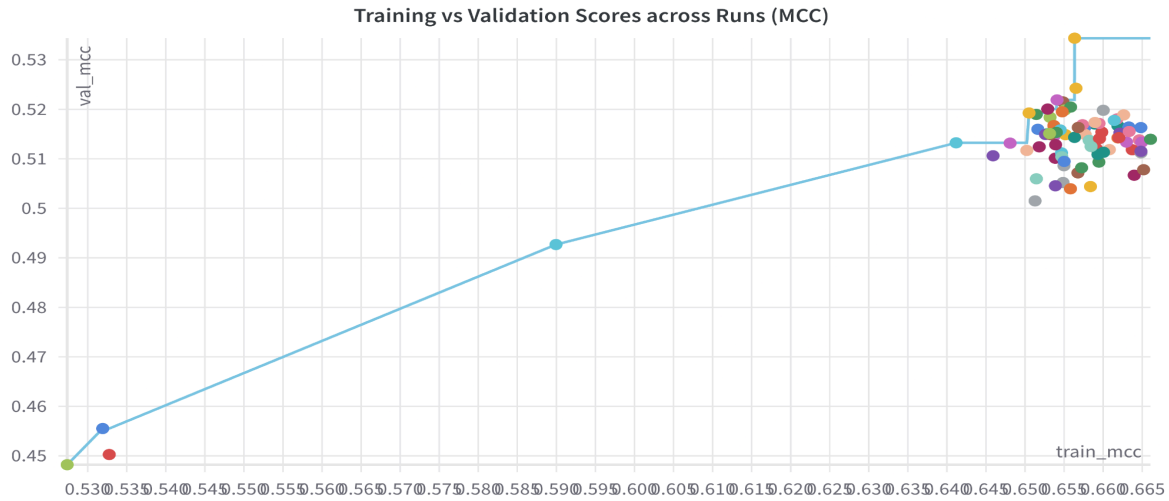
- **Evaluation Metric:**
    - The results of metric type that the sweep is considering when selecting hyperparameter values in it's search strategy
- **Termination Criteria**

We start with a very broad search space, consisting of value ranges defined for every relevant hyperparameter. After some promising parameter values emerge, we start a new sweep with a more narrow search space.

Broad Sweep

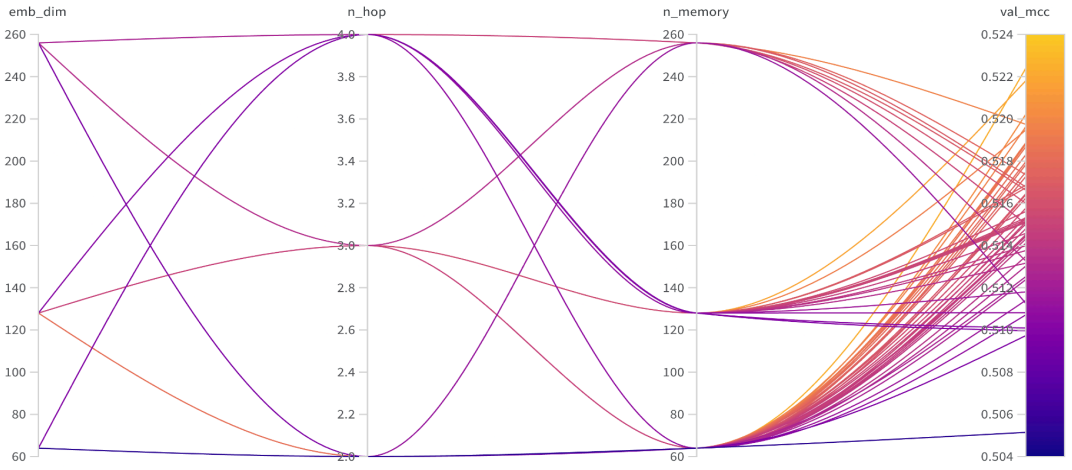


The figure above consists of paths combining different hyperparameter values. The ends of the paths then point towards the resulting performance metric value of the trained model with those value combinations. The Bayesian Optimization strategy clearly fixates on the transformation matrix method of predicting synergy scores (which was the default option in GraphSynergy), as well as slight emphasis on higher embedding dimension sizes. However, it is important to point out that, any given value combination still fluctuates in a very narrow performance range, mostly around a Matthew's Correlation Coefficient (MCC) from 0.50 to 0.52, meaning any better combination still yields only marginal improvements. This is also reflected in the following figure, which maps out the training score against the validation score of each hyperparameter combination, to distinguish differences in generalization capabilities.



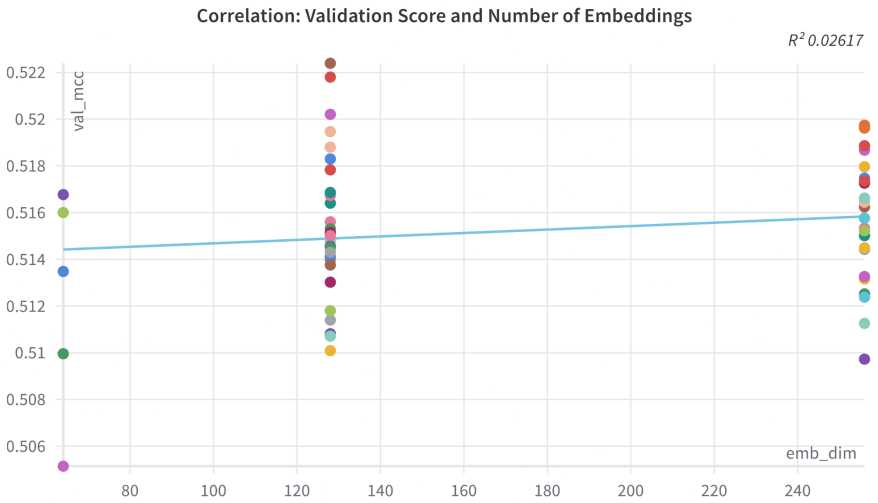
Here, we can clearly see a heavy clustering of combinations in a single generalization range, which would reflect a very weak correlation between any single hyperparameter value and generalization capabilities.

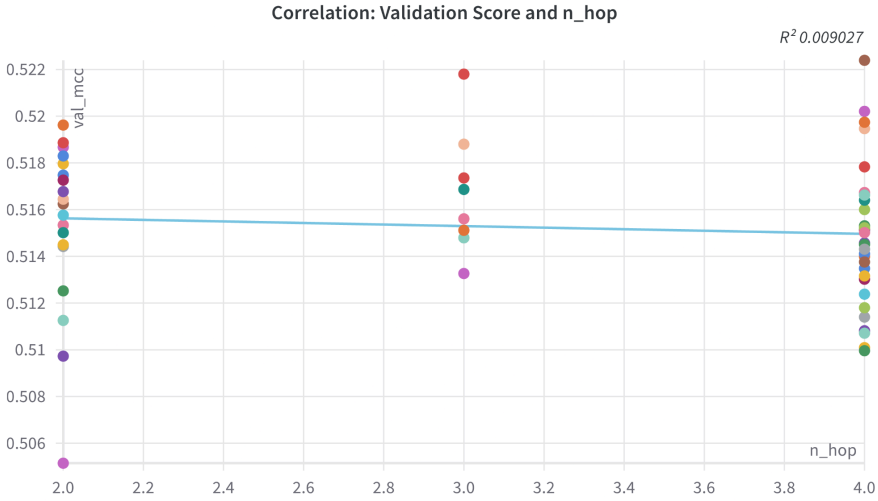
Narrow Sweep

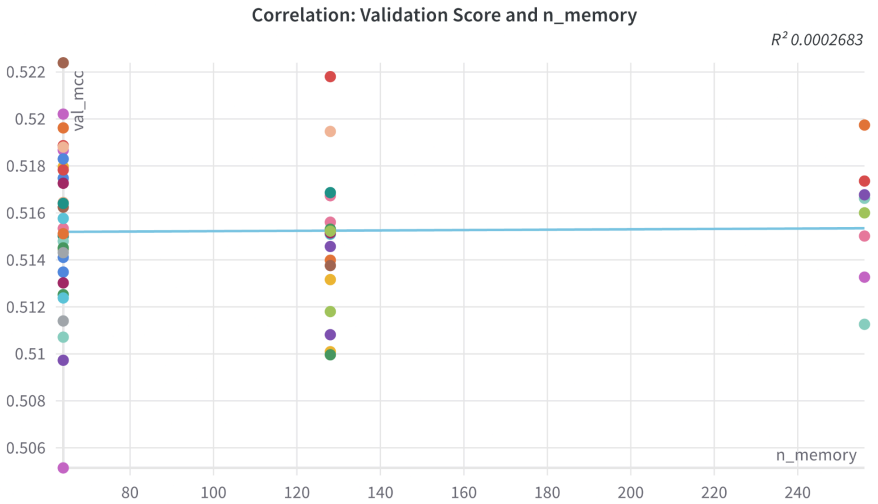


This next sweep narrows the search space, by optimizing the three parameters that are specific to GraphSynergy (embedding_dim, n_hop, n_memory). The paths in the narrowed sweep, still fall in the same performance range as before, again suggesting no clear correlations between higher/lower parameter values and improved performance. This is observed clearer with the given scatter plots in the correlations section below.

Correlations







For all three parameter types, it can clearly be seen that there is a large overlap in the performance values that different discrete parameter values fall on, showing low correlation between the prediction performance and hyperparameter values. This is also evident by the given very R^2 values.

Key Takeaways

The results of our sweeps suggest, that while there is some improvements to be had here depending on the chosen hyperparameter values, these improvements can be classified as diminishing returns, as they are often only marginal and in case they require the selection of larger hyperparameter values, they often significantly impact computational efficiency when training an inferencing the model.

As a result, we concluded that the default hyperparameters are already optimal, and used those in any and all subsequent training and testing.

Analysis of The PPI-Graph and its Topology

In graph-based machine learning models such as GraphSynergy, the underlying graph plays a key role in determining model performance. In the case of GraphSynergy, the graph in question represents a *Protein-Protein-Interaction Network* and maps out the relationships and interactions between various proteins that are found in tissues and cell-lines. This provides the model with important contextual information about protein interactions to train on.

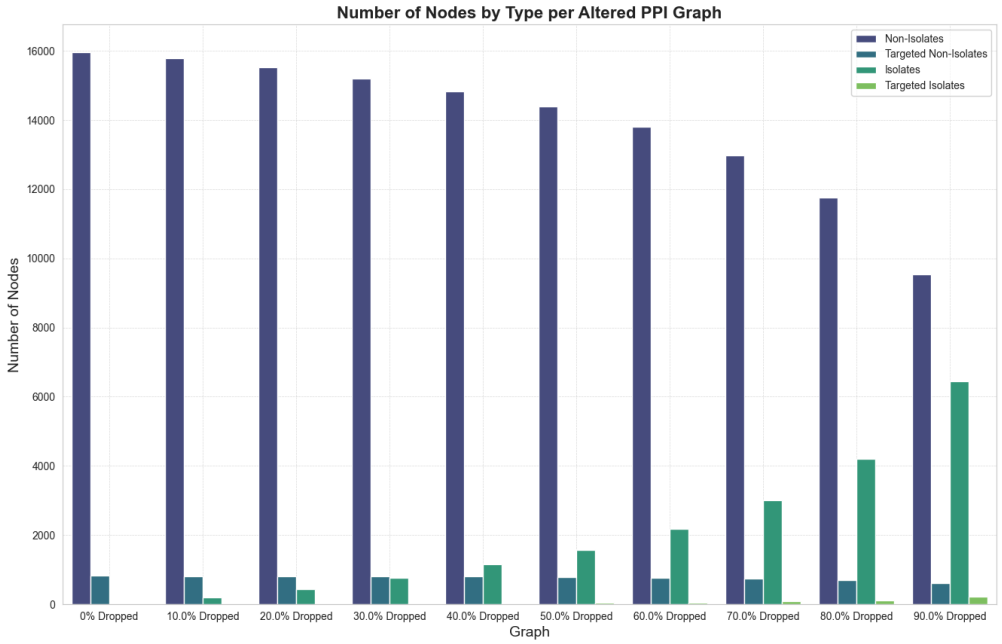
Given that one of the main purposes of this study is to understand to what extent the proposed neural network is reliant on the graph information, it is important to analyse the PPI-Graph structure, as this might provide early insights into the models resilience and robustness capabilities to manipulations of the graph information. We were particularly interested in how severing connections in the graph would affect the overall connectivity and access to information across the PPI-Graph, so in our analysis we also included statistics of graphs after different levels of random edge removals.

We also did a separate investigation into the statistics of neighborhood for each protein and which neighbors are selected. Additionally we looked at how the neighbors are identified and selected and found out there is a bug in the algorithm that causes the wrong proteins to be selected.

Directly Targeted and Isolated Nodes

The PPI-Graph consists of a large network of various protein interactions, however not every represented interaction is equally important to the model. During the training process, the model primarily focuses on protein interactions that are directly or indirectly targeted. The surrounding area of these directly targeted proteins is considered depending on the drug combinations, the cell line as well as hyperparameters that determine how far into the graph the model should read into. To better distinguish this, we introduced the following classifications:

- Targeted proteins – Proteins directly associated with drugs that are present in the cell lines
- Isolates – Proteins in the PPI-Graph that do not have any interactions (connections) with any other protein

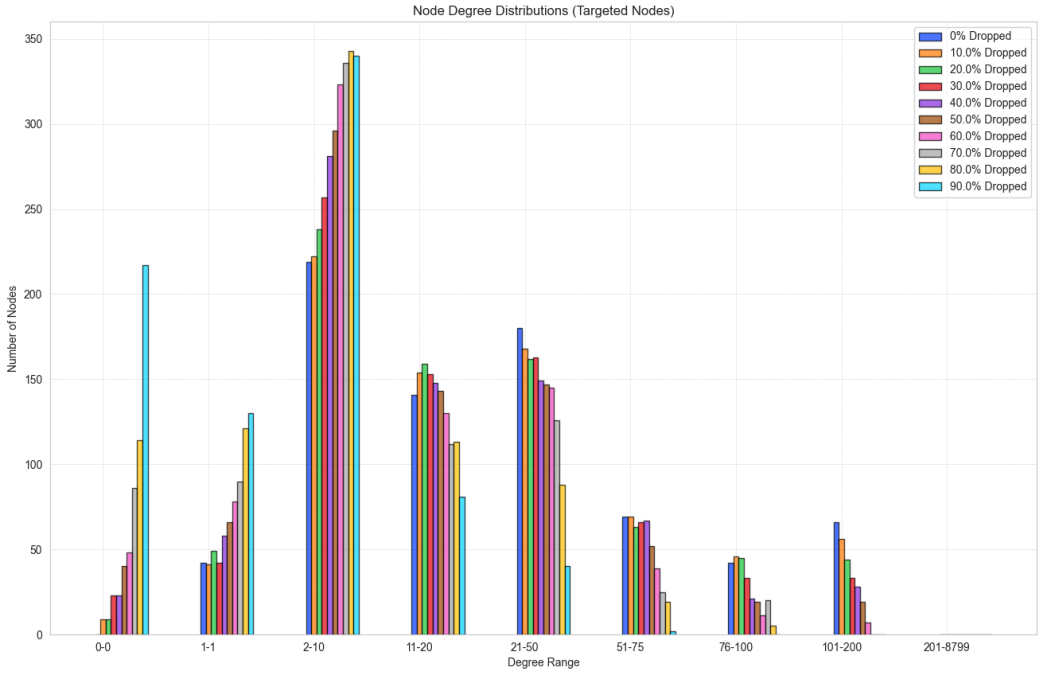


The figure above, shows the frequency counts of nodes by the classifications we introduced for different levels of randomly disconnected edges. In the default PPI-Graph, there are no Isolates to speak of and a very small portion of all nodes consist of directly targeted proteins. As edges are being removed, the change in the number of isolates grows exponentially, however, the growth in the number of

targeted isolated nodes does not reflect the same acceleration and increases much slower. The increase becomes more pronounced around 60% of edges being, removed, which might hint towards a relatively good resilience for the model against removing edges from the PPI-Graph

Node Degree Distribution of Targeted Nodes

For a more detailed analysis, we also looked at the node degrees of targeted proteins. The Degree of a node, stands for the number of direct connections a given node has. Knowing the degree of a node, helps us understand the information that is available for aggregation around the immediate vicinity of a node.



The histogram above, shows frequency bins for different node degree ranges and their distributions for different levels of edge removals. The node degrees in the initial PPI-Graph resemble a normal distribution with high variance around the 2-10 degree bin. This means there is a modest to high number of connections for most nodes, the distribution tends to skew to the left as edges are being removed, however the graph seems to maintain modest degrees for most nodes, with the number of isolated nodes starting to grow rapidly around 60% of edges being removed. This again reinforces the analysis of the previous figure and depending on

the number of hops and number of neighbours being aggregated at each hop the model could be able to aggregate enough information for good generalization through initial phases of edge removal.

Clustering Coefficients

The clustering coefficient of a graph measures to what extent nodes tend to form-up in tightly connected clusters. It ranges in a fixed scale from 0 to 1, with 0 meaning none of the neighbors are connected and 1 meaning all neighbours of a node are also connected with each other. It provides insights into how interconnected a node's neighbours are. There are different implications for a graph depending on its clustering coefficient.

- Depending on the number of aggregations and hops, in **a highly clustered graph** the model might aggregate the same node information repeatedly. In that case, the model might put too much emphasis on localized protein interactions and fail to generalize to drug combinations targeting larger areas in the cell lines.
- Adversely, in a graph with **lower clustering** and hence containing more sparsely connected neighbourhoods of nodes the model might struggle in learning protein interactions in local protein structures. Depending on the amount of aggregated node information, this might lead to lower overall accuracy, especially for drug effects in smaller areas.

Global Clustering Coefficients with Edge Removal

	Graph	Global Clustering Coefficient
0	0% Dropped	0.2025
1	10.0% Dropped	0.1769
2	20.0% Dropped	0.1453
3	30.0% Dropped	0.1295
4	40.0% Dropped	0.1006
5	50.0% Dropped	0.0783
6	60.0% Dropped	0.0567
7	70.0% Dropped	0.0343
8	80.0% Dropped	0.0195
9	90.0% Dropped	0.0059

The table above shows the global clustering coefficients with different levels of edge removal applied on the PPI-Graph. The default graph already has a low to moderate clustering and as expected, the global clustering coefficient drops in increasing rates as more edges are being removed, reaching near 0 when 90% of edges are removed.

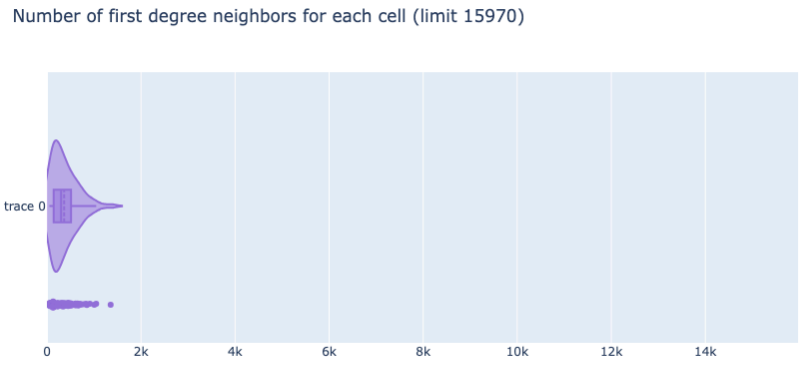
This reflects the model's fragility in local connectivity, and depending on the aggregation scheme as well as the drug combination effects in the dataset, the model might struggle in making accurate predictions in case the drug effects put emphasis on more localized protein interactions.

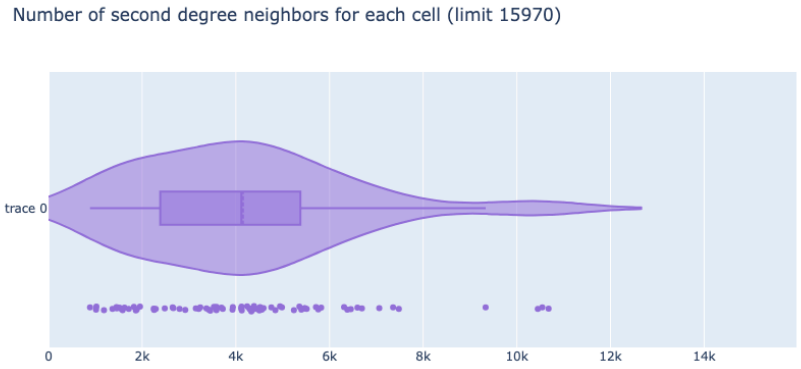
Neighborhood Analysis

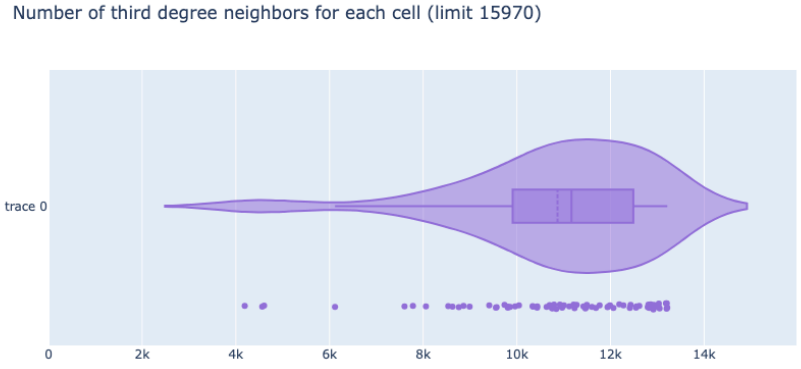
In their paper Yang et al. not only use the direct neighbors, but also further distant neighbors. They characterize neighbors based on the number of hops a neighbor is away from the origin. In the default configuration 3 hops are used as the limit considered for the neighborhood. There is also a limit of 128 proteins that are selected for every hop.

For each hop we analyzed how many neighbors there are for each protein while separating cell-line and drug-target proteins. To get a visual understanding of these numbers we created some violin plots.

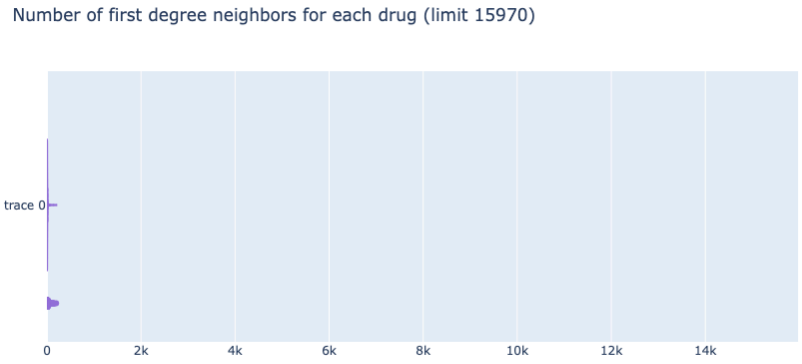
First we take a look at the cell-line proteins. When only considering the first degree neighbors which only take one hop, we see that many proteins already have a lot more neighbors than the 128 limit. This gets even more extreme for the second and third hop where all proteins have a lot more than 128 neighbors and many even multiple of thousands. This calls into question how helpful it is to consider these neighbors, if most of the proteins from the PPI graph are part of the set as there are only 15970 proteins in the graph overall.

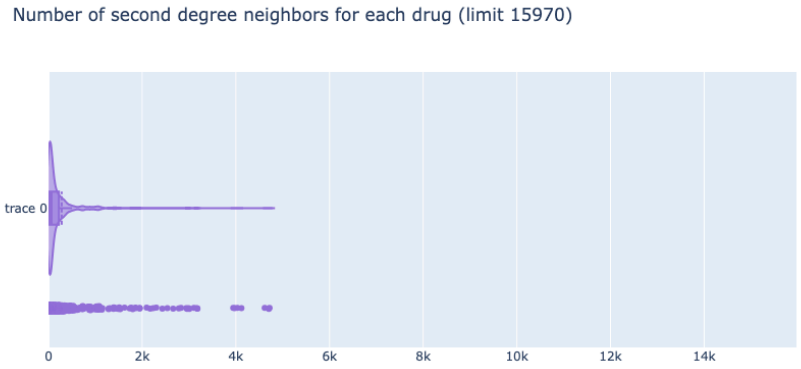


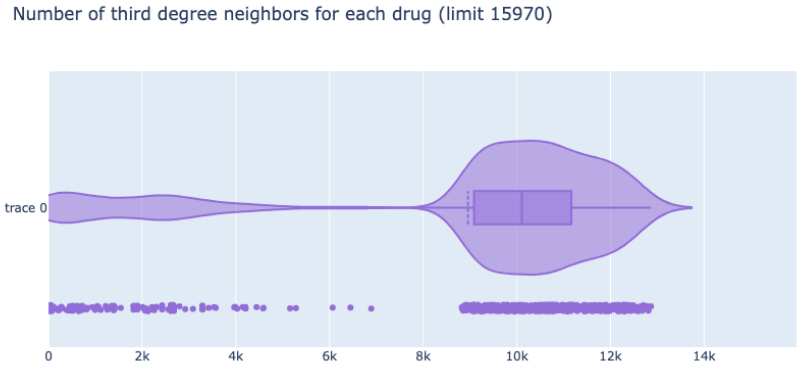




When taking a look at the same statistics for the drug-target proteins we get a similar distribution of values. For the first hop neighbors we see lower values for each protein with only a few proteins that have more than the 128 limit. For the second hop neighbors we also see a big increase in the amount of neighbors with many proteins over the 128 limit, but it is a bit less extreme as numbers of first hop neighbors are a lot lower. Ultimately for the third hop neighbors we see very high numbers for most proteins similar to the cell-line proteins. In the case of the drug-target proteins it seems to make more sense to look at neighbors as there are overall fewer, but as soon as the second hop neighbors the selected 128 are only a small subset in most cases.







When we looked at the function that created for each protein lists of neighbors with a separate list for each hop, we found a bug. In the loop of line 174-175 (link) when the neighbors of the next hop are gathered, the authors forgot to remove proteins from previous hops.

```
160     def get_neighbor_set(self, items, item_target_dict):
161         print('constructing neighbor set ...')
162
163         neighbor_set = collections.defaultdict(list)
164         for item in items:
165             for hop in range(self.n_hop):
166                 # use the target directly
167                 if hop == 0:
168                     replace = len(item_target_dict[item]) < self.n_memory
169                     target_list = list(np.random.choice(item_target_dict[item], size=self.n_memory, replace=replace))
170                 else:
171                     # use the last one to find k+1 hop neighbors
172                     origin_nodes = neighbor_set[item][-1]
173                     neighbors = []
174                     for node in origin_nodes:
175                         neighbors += self.graph.neighbors(node)
176                     # sample
177                     replace = len(neighbors) < self.n_memory
178                     target_list = list(np.random.choice(neighbors, size=self.n_memory, replace=replace))
179
180                 neighbor_set[item].append(target_list)
181
182         return neighbor_set
```

We fixed this bug by keeping a list of old neighbors to remove them when collecting the neighbors of the next hop in line 213 (link).

```
199                 neighbor_set = collections.defaultdict(list)
200                 for item in items:
201                     neighbors_old = set()
202                     for hop in range(self.n_hop):
203                         # use the target directly
204                         if hop == 0:
205                             replace = len(item_target_dict[item]) < self.n_memory
206                             target_list = list(np.random.choice(item_target_dict[item], size=self.n_memory, replace=replac
207                         else:
208                             # use the last one to find k+1 hop neighbors
209                             origin_nodes = neighbor_set[item][-1]
210                             neighbors = []
211                             for node in origin_nodes:
212                                 neighbors += self.graph.neighbors(node)
213                             neighbors = set(neighbors).difference(neighbors_old)
214                             neighbors_old = neighbors_old.union(neighbors)
215                             neighbors = list(neighbors)
216                             # sample
217                             replace = len(neighbors) < self.n_memory
218                             target_list = list(np.random.choice(neighbors, size=self.n_memory, replace=replace))
219
220                         neighbor_set[item].append(target_list)
```

PPI-Graph Manipulation

In this chapter, we explore various manipulation techniques on the graph that alter its structure, to see how it affects its performance.

Random Edge Shuffling

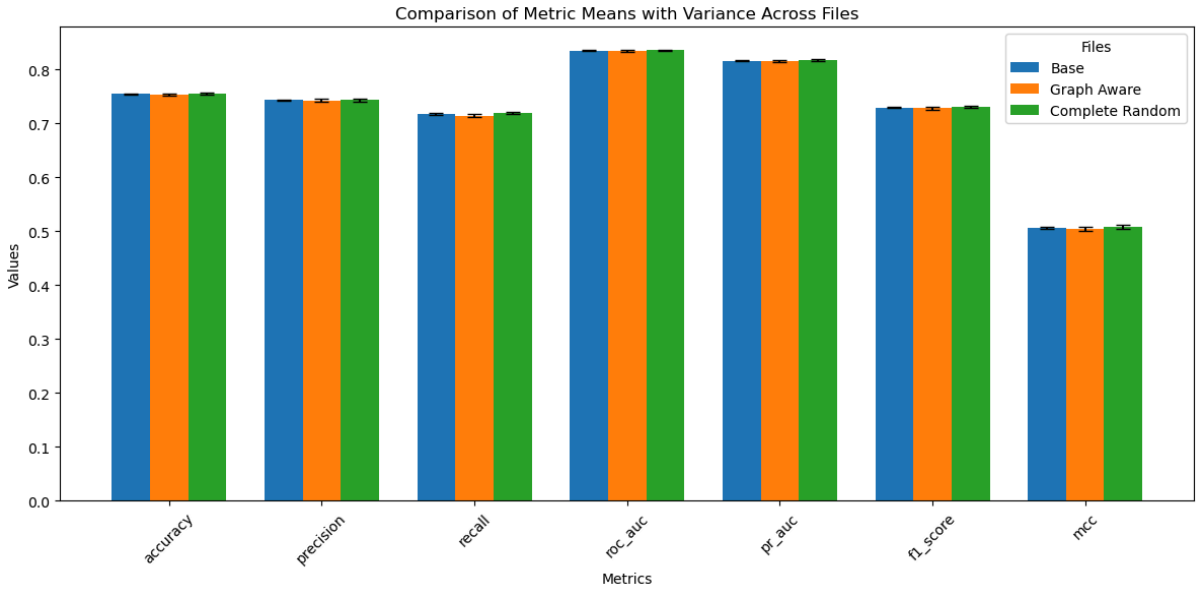
1) Graph Aware Edge Shuffling

For this shuffling method, a node's edges are picked at random from the edges it already has. in other words; a node will not have new edges, but some are removed. Which edges are removed and how many are completely random.

2) Complete Random Edge Shuffling

For this method, a node's edges will be completely random. When fetching a node’s edges, it simply picks a complete random set of nodes in the graph. it will remove all pre-existing edges and generate new ones at random.

Experimental Results for Random Edge Shuffling



The plot shows the model's performance of the two different edge shuffling methods compared to no shuffling (base). The findings are very interesting as the shuffling has no statistical significant impact on the performance. This indicates that the edges are doing nothing to improve the model.

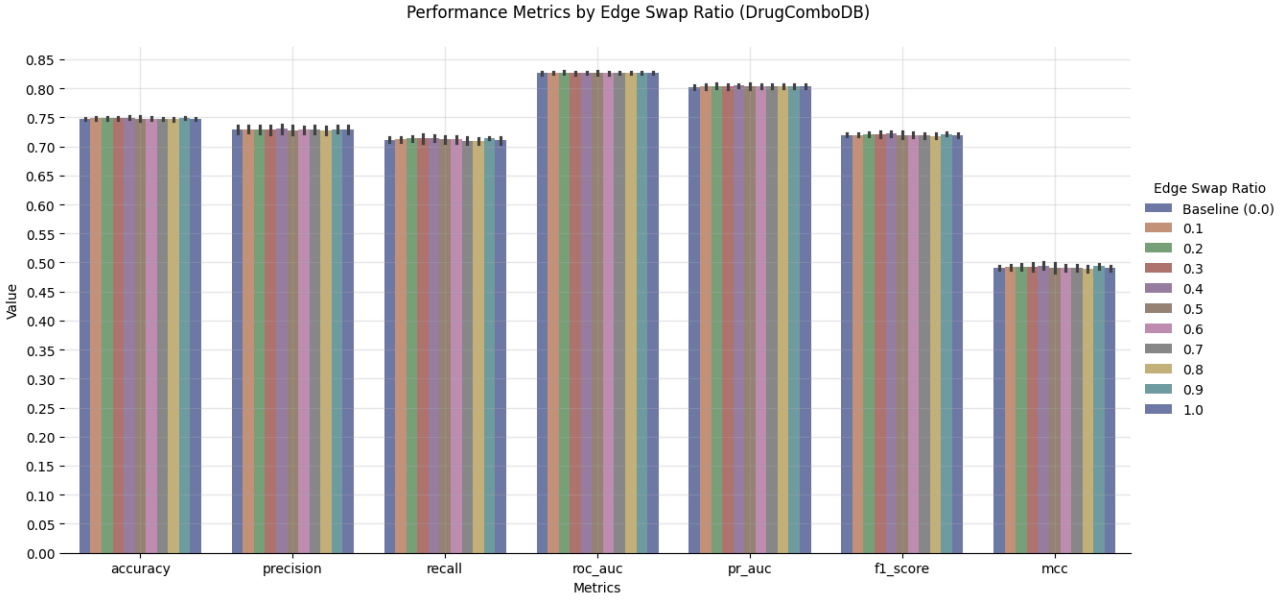
3) Degree Preserving Edge Shuffling

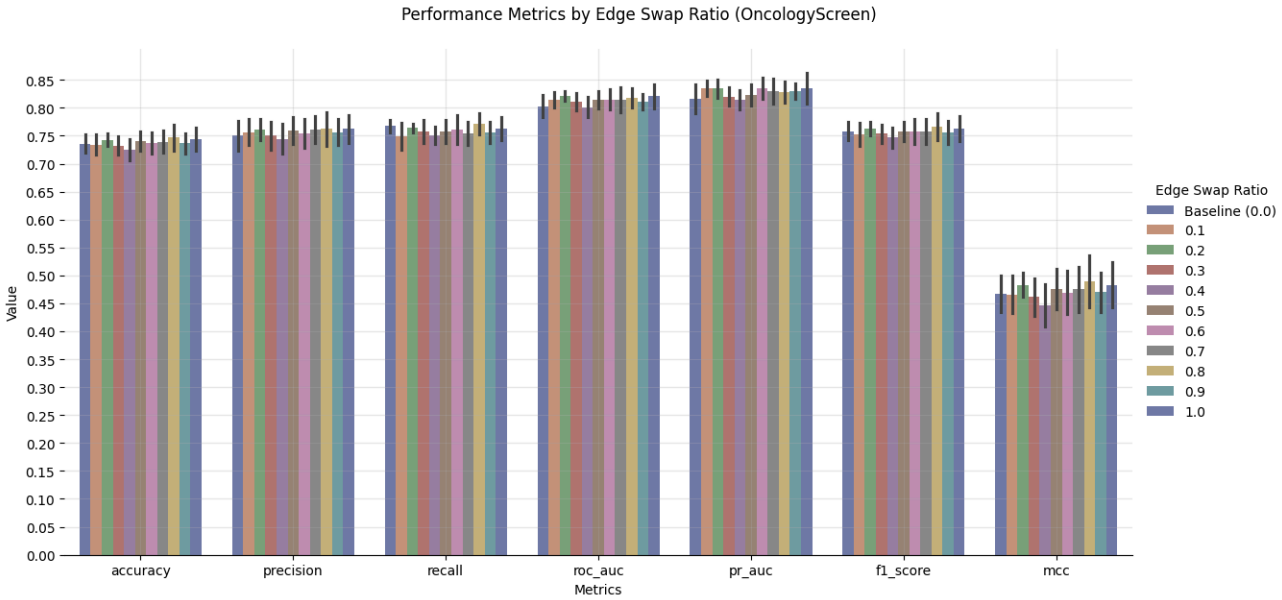
Another edge shuffling approach shuffles the edges at random, however preserves the original node degrees, meaning the number of neighbors a node has remains the same but with different neighbours. This is achieved through an approach called "*double edge swap*", which selects two edges at random that connect disjoint nodes, disconnects them and connects nodes from the disjoint sets:

e. g., given 2 edges (u, v) and (x, y) are removed and new edges (u, x) and (v, y).

In our test results, we compare cross validation results for models that were trained on graphs with incrementally increased degree preserving edge shuffling rates (10% of all edges, 20% of all edges, etc.).

Experimental Results for Degree Preserving Edge Shuffling





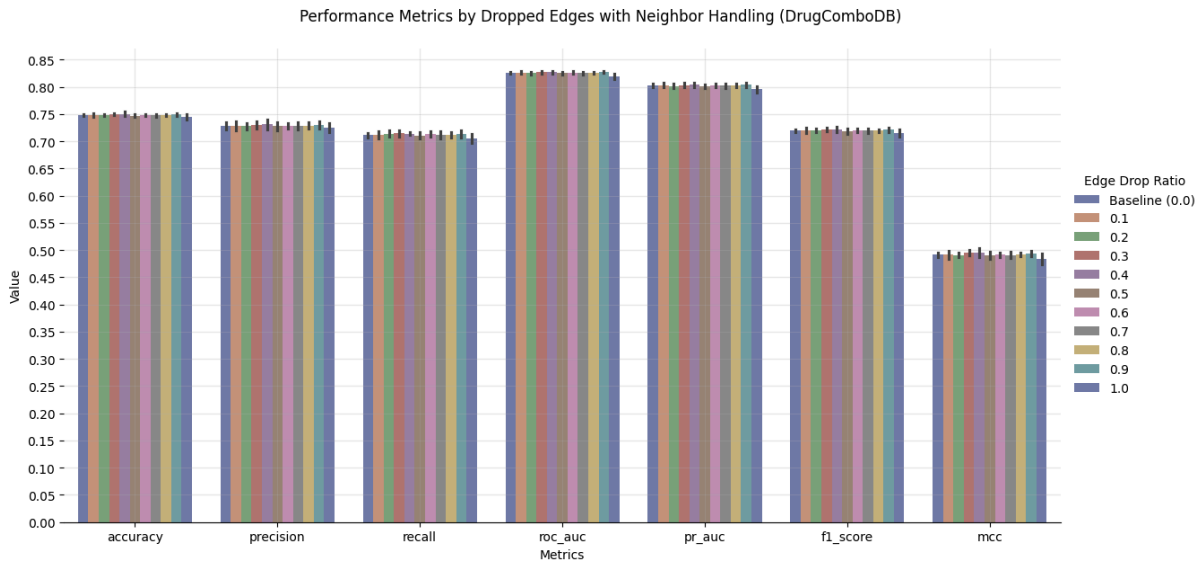
The plots compare metric scores for increasing edge shuffling rates, with the bar showing the mean prediction score and the error bars showing the standard deviation across test sets. For the results on the DrugComboDB dataset, it can be clearly seen that all scores are nearly identical or in the case of the OncologyScreen dataset fall within the margins of error of each other, suggesting that changing node connections while preserving their counts has no clearly distinguishable effect on model performance. The higher standard deviation present in the results for OncologyScreen is likely attributed to its low data volume in comparison with DrugComboDB.

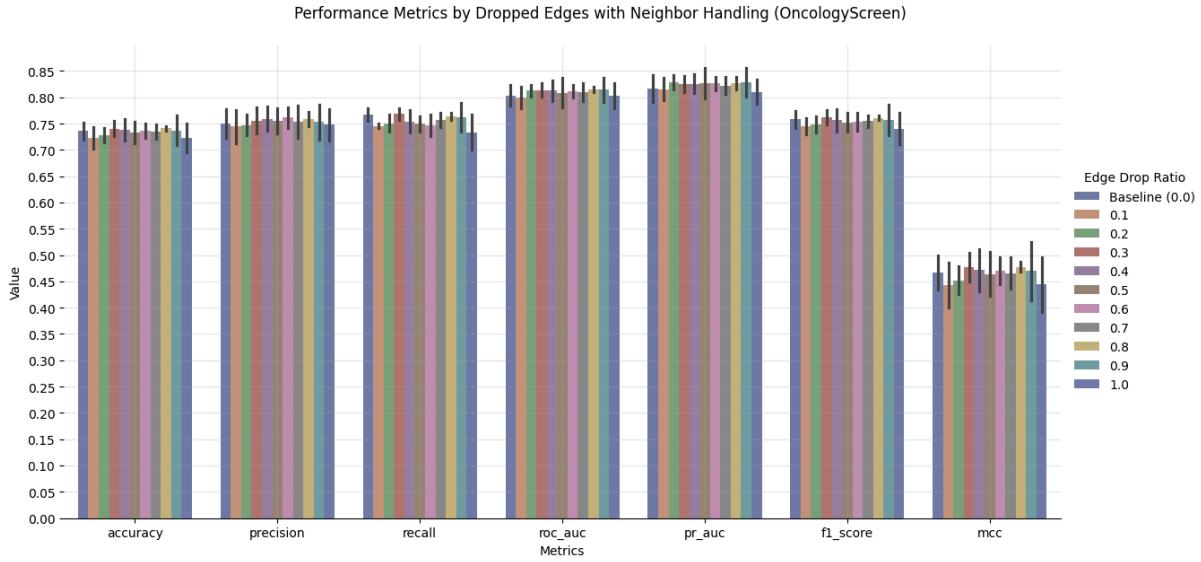
Random Edge Removal

Along with shuffling, another manipulation technique we investigated was to randomly and incrementally severe connections from the graph. The results of which demonstrate if the loss of protein interaction information is detrimental to the models inference capabilities.

In this technique, we built the PPI-Graph normally and as with the degree preserving edge shuffling, incrementally and randomly removed edges. One aspect that wasn't accounted for in the original implementation, was the correct handling of neighbor aggregation for isolated nodes, as they do not appear. We handled this by aggregating the node itself as its neighbours if it didn't have any neighbours of its own. This ensures consistent behaviour for all isolated nodes and is inline with the original neighbor aggregation scheme, that allows for the aggregation of duplicate neighbours of a node if the node doesn't have enough neighbors to meet the n_memory hyperparameter threshold.

Experimental Results of Random Edge Removal





Similarly to the results of the Degree Preserving Edge Shuffling, each model trained on graphs with edges removed at different severity levels, perform nearly identical for DrugComboDB and slightly more volatile but at similar levels for OncologyScreen. One consistency of note for both datasets is that, having all edges removed from the graph results in a slight decrease in performance across all metrics, however the loss is still small enough to fall into the margin of error.

The results across all manipulation techniques demonstrate that the model is surprisingly resilient to both small and major manipulations of the graph topology i. e. manipulations of the protein interaction information. The results seem to strongly imply that the PPI-Graph is not critical for the neural networks performance and that it might be relying on other features rather than the modelled protein interactions.

Drug-Synergy Correlation

Because the edges in the graph provide no improvement to the model's performance, we want to see if we can find some correlation between different metrics of two drugs and their synergy score. If there are any, then we can use these metrics to assign weights and biases to the edges.

Metrics

We looked all over the internet for different drug databases that can provide information about the drugs. There were plenty, but not many of them provided a csv (or similar) of all drugs and their attributes, nor an API. We did however find some.

For the dataset of roughly 70 000 drug combinations, we were able to find metrics for enough drugs to get roughly 50 000 combinations. The metrics we found include:

Molecular Weight

To compare two drugs we looked at their:

- Combined weight
- Difference in weight
- The ratio between the weights
- The geometric mean of the weights

Smiles String

To compare two drugs we used the following:

- Tanimoto coefficient
- Cosine similarity
- Lingo similarity

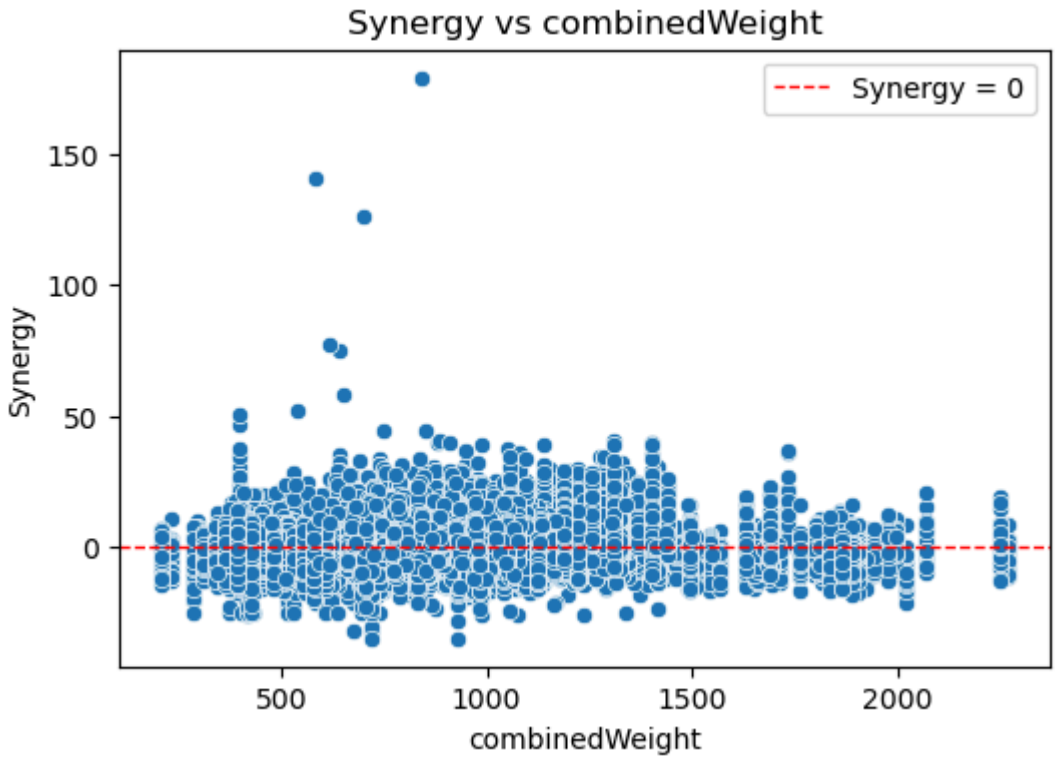
Proteins

To compare two drugs, we looked at how many proteins they have in common

Correlation Testing

For all the metrics mentioned above and synergy scores, we created scatter plots and ran a correlation test. For the correlation we looked at both the precise score and the score as a binary “synergistic or not synergistic” (score of more or less than 0). The correlation coefficient used is Pearson's R. The results are as follows:

Combined Weight



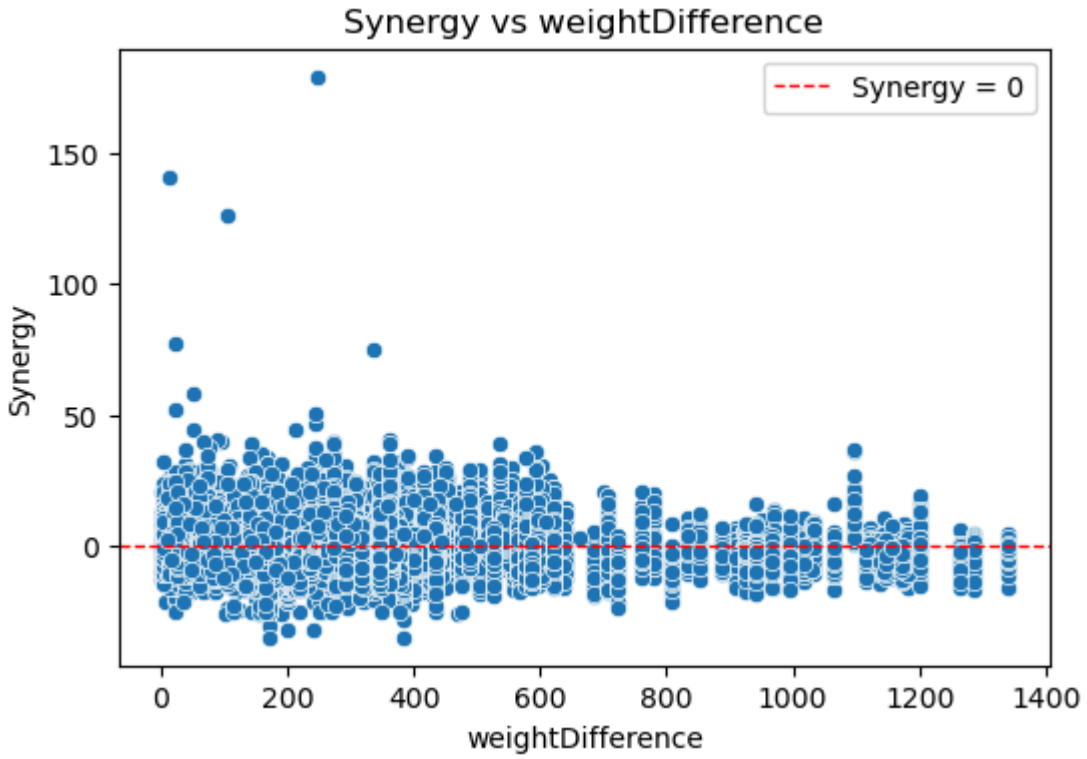
Non-binary

- Correlation Coefficient: 0.11
- P-Value: 3.83e-136
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Binary

- Correlation Coefficient: 0.09
- P-Value: 1.08e-86
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Weight Difference



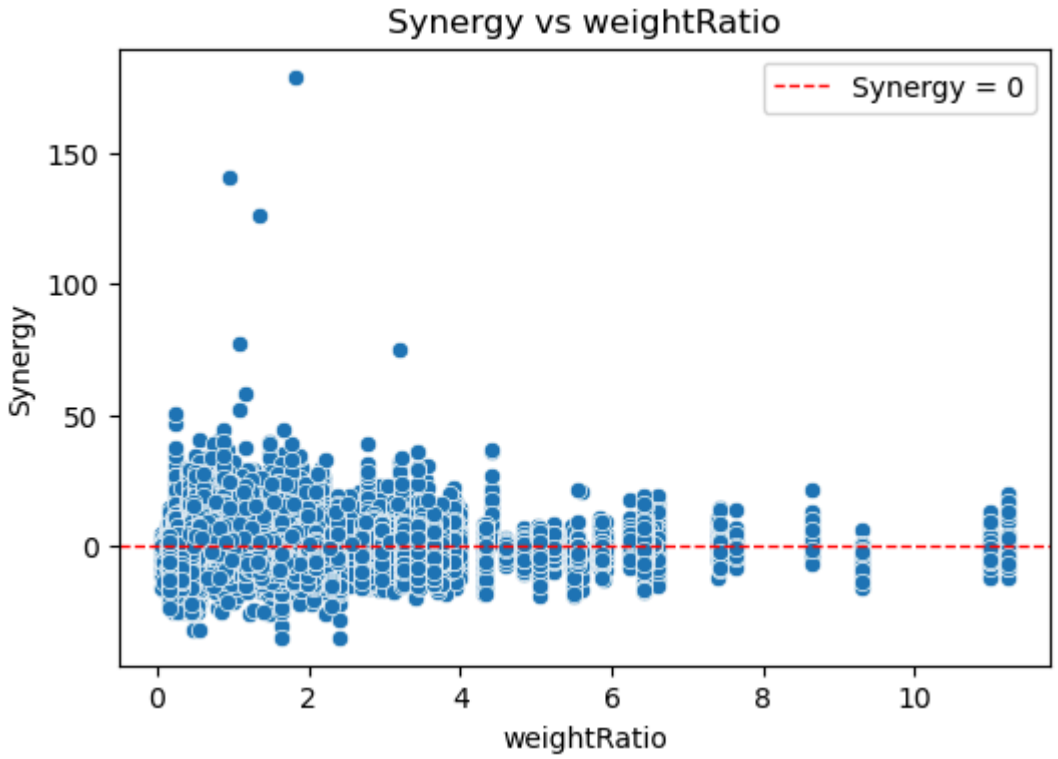
Non-binary

- Correlation Coefficient: -0.4
- P-Value: 2.91e-0.19
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Binary

- Correlation Coefficient: -0.04
- P-Value: 1.70e-19
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Weight Ratio



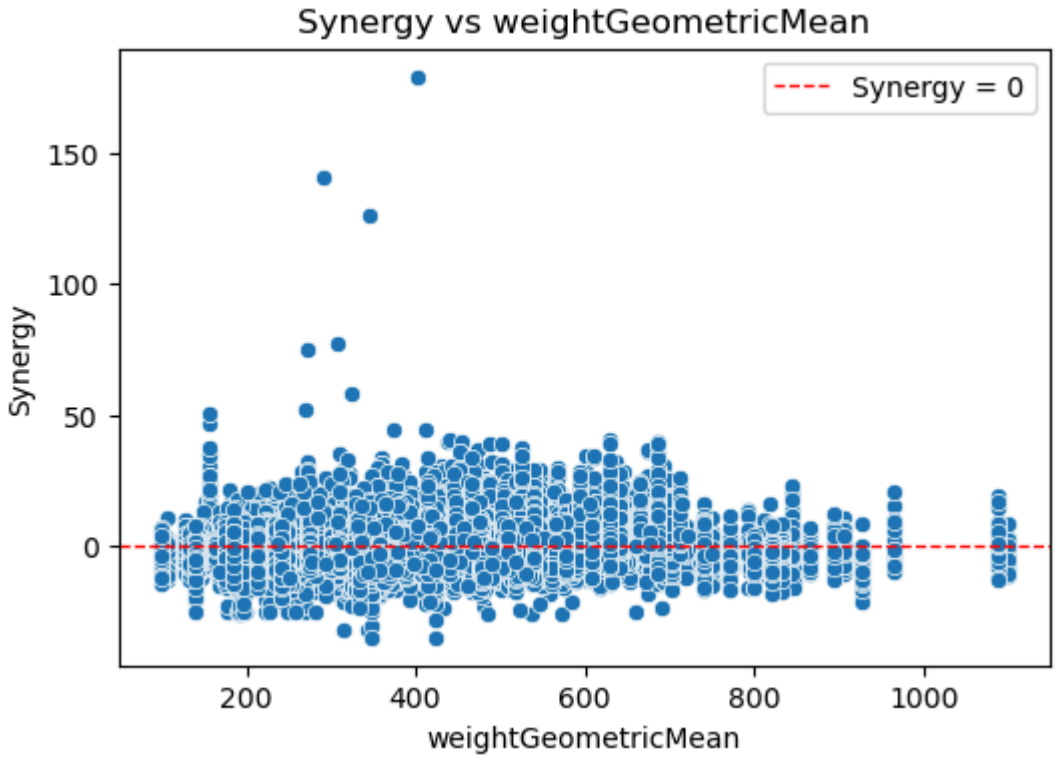
Non-Binary

- Correlation Coefficient: -0.2
- P-Value: 2.91e-0.4
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Binary

- Correlation Coefficient: -0.01
- P-Value: 1.91e-02
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Weight Geometric Mean



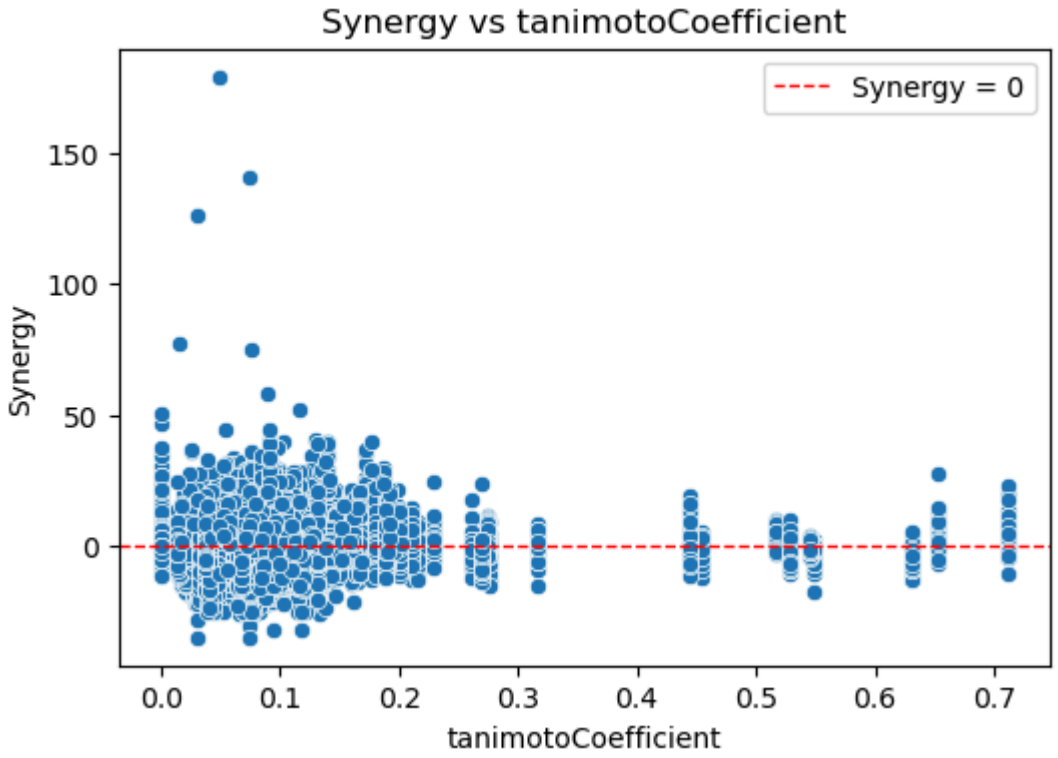
Non-binary

- Correlation Coefficient: -0.14
- P-Value: 9.55e-220
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Binary

- Correlation Coefficient: -0.11
- P-Value: 2.71e-145
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Smiles Tanimoto Coefficient



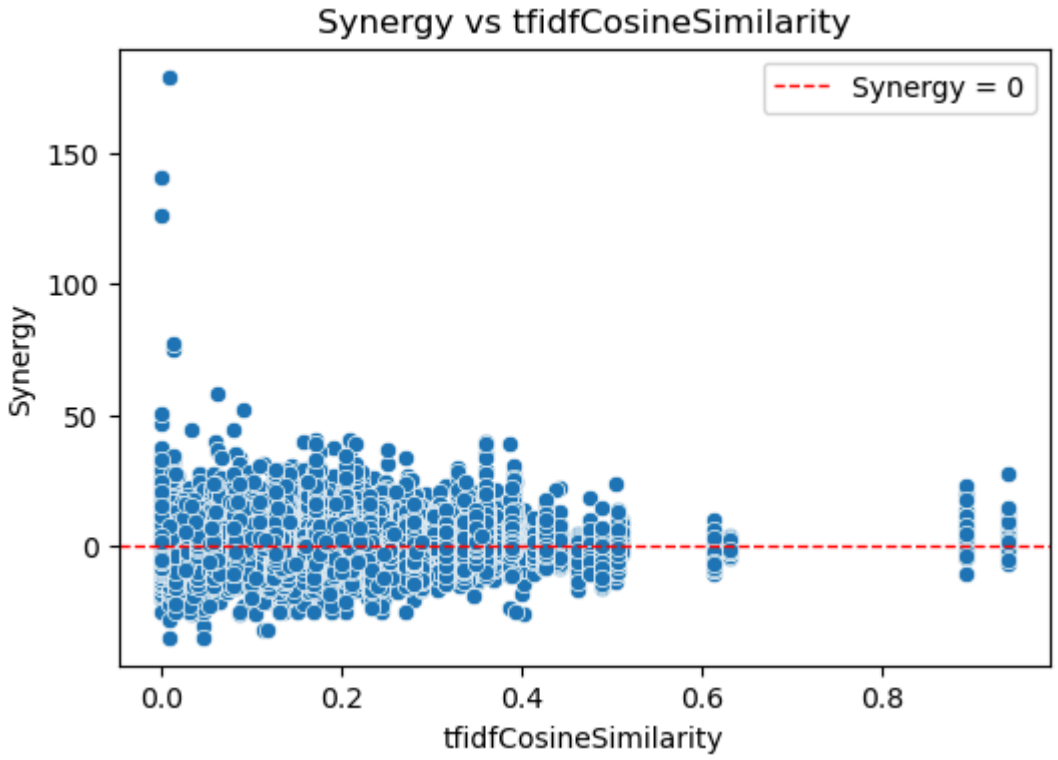
Non-binary

- Correlation Coefficient: -0.05
- P-Value: 3.75e-30
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Binary

- Correlation Coefficient: -0.04
- P-Value: 6.33e-18
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Smiles Cosine Similarity



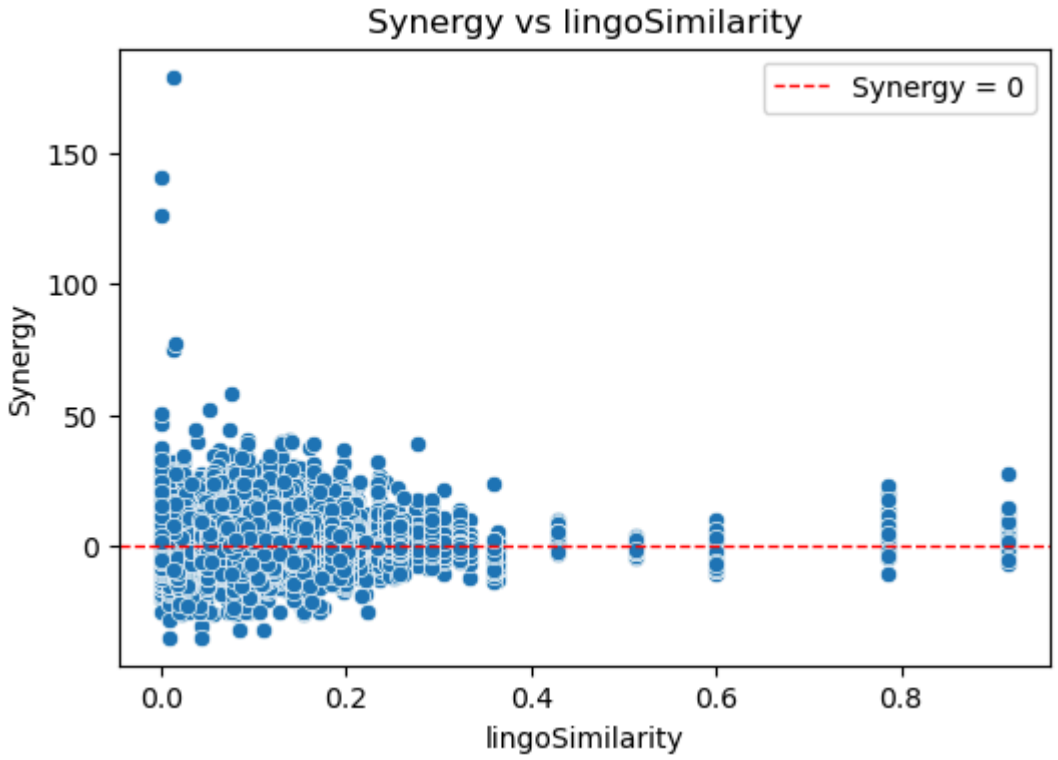
Non-binary

- Correlation Coefficient: -0.1
- P-Value: 9.43e-113
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Binary

- Correlation Coefficient: -0.09
- P-Value: 1.15e-93
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Smiles Lingo Similarity



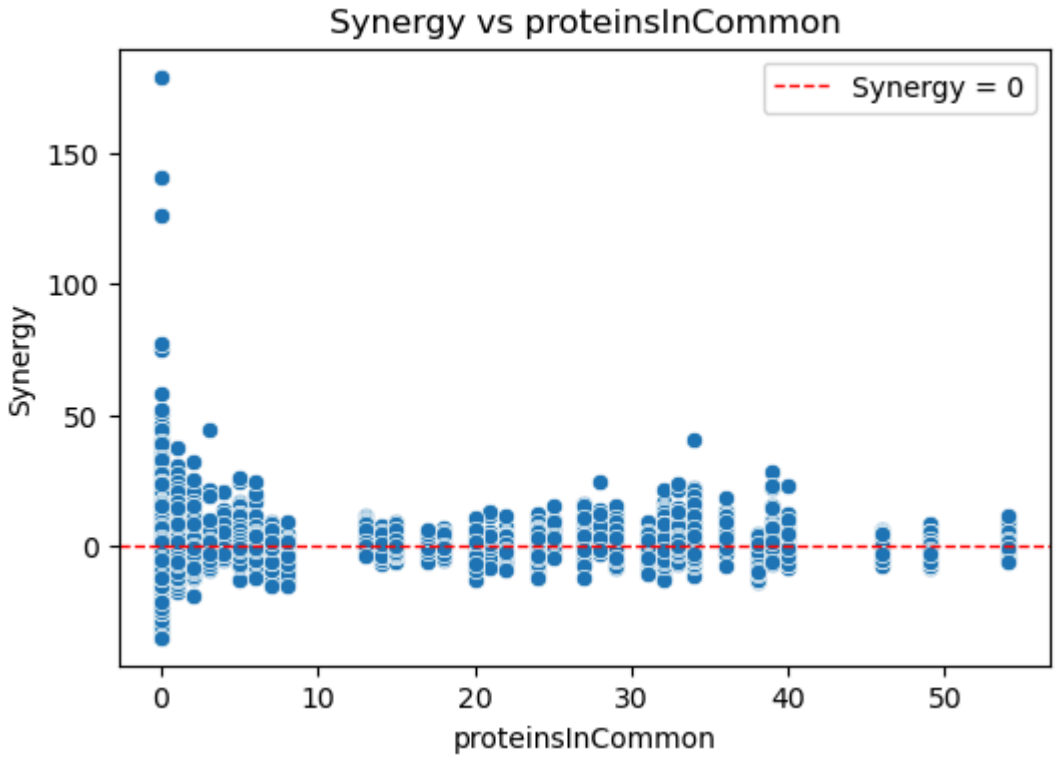
Non-binary

- Correlation Coefficient: -0.09
- P-Value: 2.81e-99
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Binary

- Correlation Coefficient: -0.09
- P-Value: 8.1e-90
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Proteins in common



Non-binary

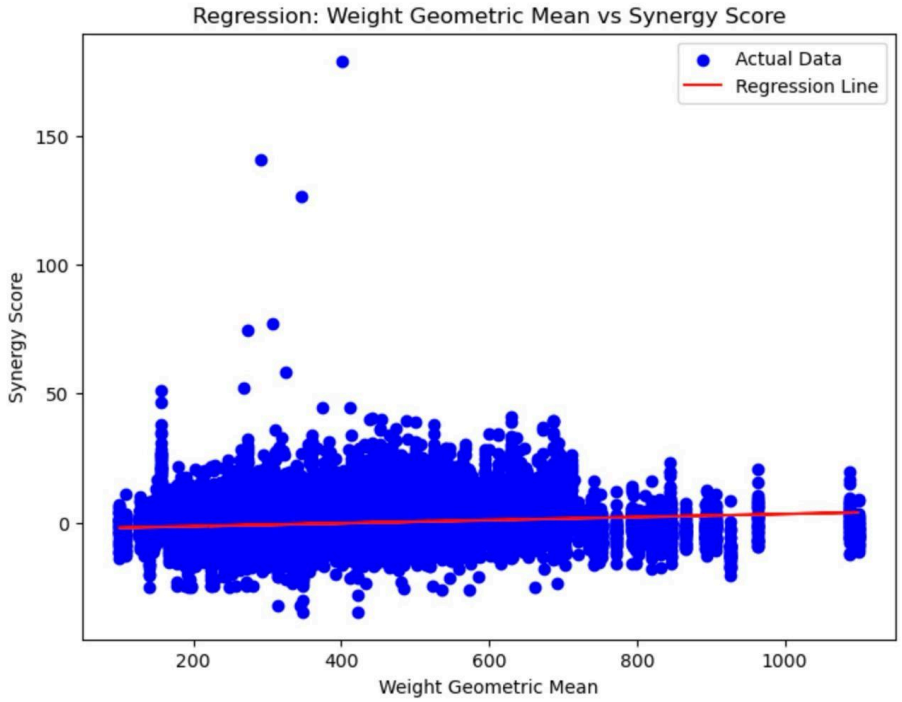
- Correlation Coefficient: -0.03
- P-Value: 1.35e-14
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Binary

- Correlation Coefficient: -0.09
- P-Value: 8.1e-90
- Strength of Correlation: Very weak or no correlation
- Significance: Significant

Linear regression

The most promising metric by far is geometric mean with non-binary synergy scores, with a Pearson's R at 0.14 and a P-value of 9.55e-220. To look closer at the geometric mean and its possibilities, we performed a linear regression. It gave the following results:



- R^2: 0.03
- Slope: 0.02

Using the regression model, we gave it all the geometric mean data points in the dataset and had it predict the synergy score. The predictions were converted to binary, synergistic or not synergistic (As this is what the GraphSynergy model does), and it had a success rate of ~58.4%.

Takeaways

All of the metrics have quite low correlation. The p-values are very good however, meaning the findings are highly statistically significant. The low correlations in addition to a poor R^2 and slope for the linear regression of geometric mean means that these metrics will most likely not be helpful for the model. We therefore did not explore this further.

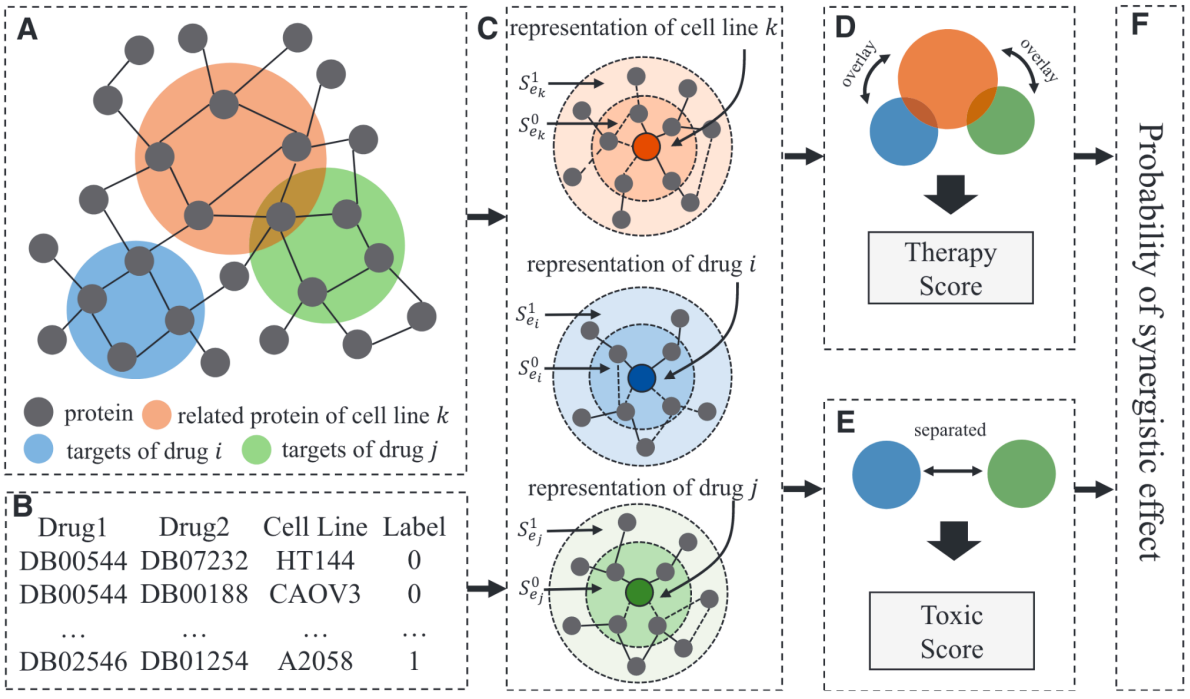
Alternative Models

The primary objective of this study was to evaluate the effectiveness of GraphSynergy, a graph-based machine learning model. Previous analyses using graph manipulation have raised questions about the importance of the PPI-Graph information and if it was truly contributing to the models performance.

Given these findings, it was essential to explore alternative approaches, leveraging both deep learning and more traditional machine learning methods that are not reliant on graph-based data structures and representations, to see:

- If simpler approaches can perform similar or better than graph-based approaches
- If non-graph based approaches demonstrate better generalization and computational efficiency
- Most importantly, whether the observed performance is truly due to the interactions represented in the PPI-Graph or just the learned drug and cell embeddings.

A recap on the GraphSynergy model:

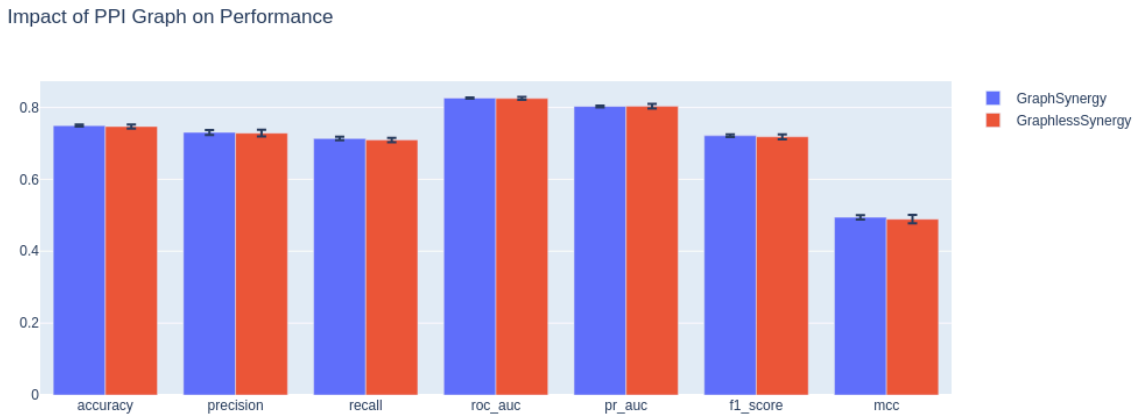


As input it takes one cell line related protein and two drugs represented by a protein target. Additionally a selection of neighbors for each protein is provided based on a protein-protein interaction (PPI) graph.

Every protein is embedded as a vector. Then a combined representation of a protein and its neighbors is calculated by a contribution of each protein neighbor to the original protein and an aggregation function over all interactions.
As a next step a therapy and a toxic score is calculated based on the combined representation of the cell and drug proteins with their neighbors.
With these scores a probability of a synergistic effect is given.

GraphlessSynergy

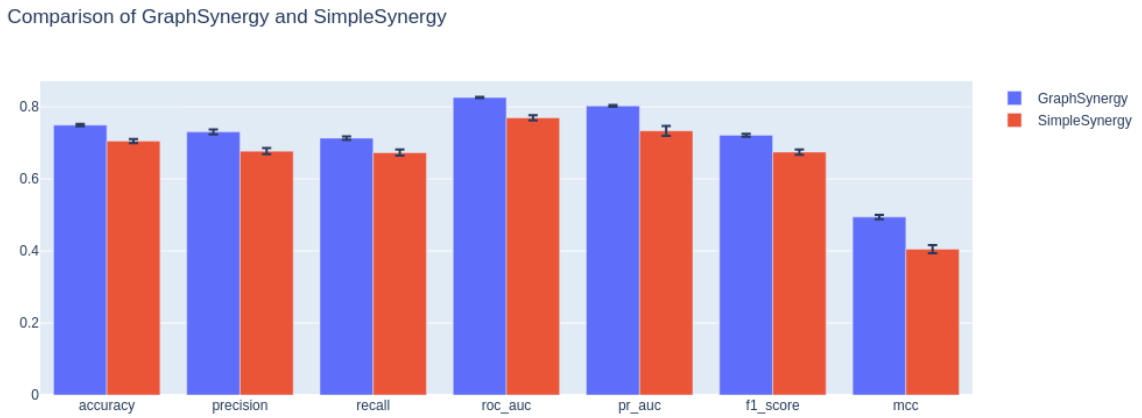
This model is supposed to be as close as possible to the original model but without access to the neighbors in the PPI graph.
To achieve this the input of the neighbors is replaced with learnable weights of the same feature dimension as the original input. This allows the network to retain its learning capabilities.
We tested the GraphlessSynergy model performance and the original model. The results were not significantly worse. This implies that the usage of the PPI graph does not have any significant impact on the performance.



SimpleSynergy

SimpleSynergy simplifies the original model by skipping the complex part of creating a combined representation of a protein and its neighbors by just taking the protein itself as input for then calculating the toxic and therapy score. Therefore the number of parameters of the model is reduced a lot.

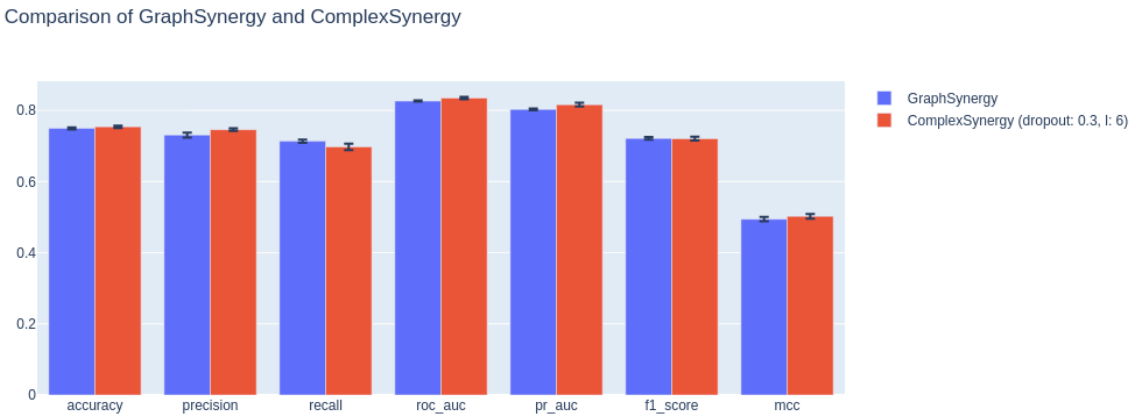
When comparing the performance we can see that reduced model capability directly correlates with significantly reduced performance scores.



ComplexSynergy

This model was made to try out the transformer architecture. It replaces the original architecture with multiple self-attention blocks also found in transformers and also only takes the proteins as input and not the neighbors. To get the final probability a fully connected layer is used with sigmoid activation function.

During testing we found that more blocks typically correlate with higher performance. In a comparison of the original model with the ComplexSynergy model with 6 layers we found that our model was significantly better for a significance level of 0.05 when looking at precision, receiver operator curve area under the curve (AUROC) and precision recall area under the curve (PRAUC). Only on the metric of recall our model was a bit worse although not significantly worse.



It is surprising that a relatively simple modern architecture can outperform a model that has access to the additional information about the neighbors in the PPI graph.

Random Forrest

Alongside the alternative deep learning approaches, we also want to explore more traditional and straightforward machine learning approaches, to see if simpler methods that are computationally much more efficient can perform on par or even better than GraphSynergy.

To demonstrate this, we decided on a straightforward Random Forest implementation. Unlike neural networks, which require extensive parameter tuning are computationally costly, especially for large networks. Random Forest provides an efficient, interpretable and robust approach to classification tasks

Experiment Setup

For a fair comparison, the random forest model was trained on the same data, implementing the same leakage free cross validation method explained previously. The training and inference involved:

1) **Feature Encoding:**

Since random forest operates on structured tabular data, instead of a graph we encoded the data using traditional approaches.

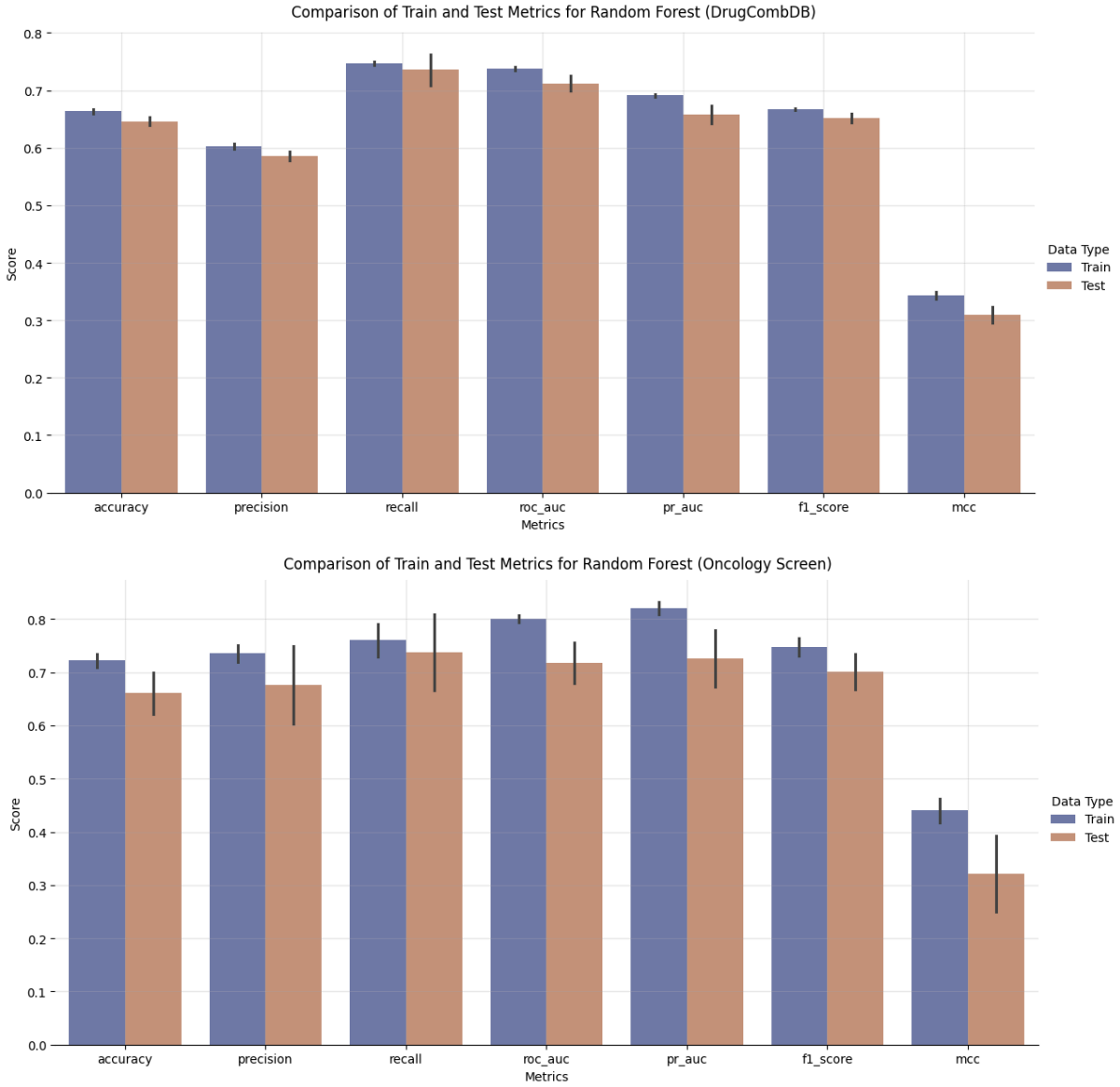
- **Multi-Hot Encoding for Drug Combinations**

- Each row in the tabular data contains a large tensor with entries for every single drug type. Entries of drug types present in the drug combination were set to ‘1’ and the remainder was set to ‘0’.
- This allows the model to capture pairwise drug interactions without introducing unnecessary sparsity, if we instead one-hot encoded each drug instance in the combination.

- **One-Hot Encoding for Cell Lines:**
  - Since cell lines appear individually and not in pairs, they were encoded with a default one-hot encoding approach.
  - Integer encoding was avoided in all encodings, to prevent the model from accidentally assuming an ordinal ordering of drugs and cell lines.

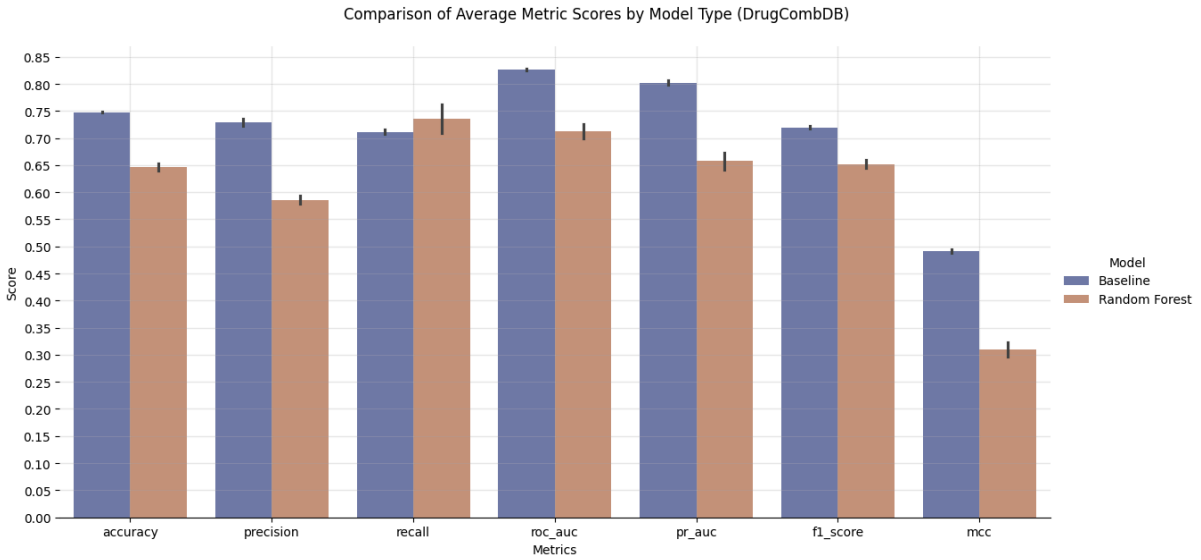
2) **Cross-Validation:**
   - The same leakage free 5-fold cross validation approach was used, this time however with 4 training, 1 validation set since, there is no hyperparameter tuning taking place.
   - Folds contain disjoint samples based on unique drug combinations, no one combination appears in another fold.

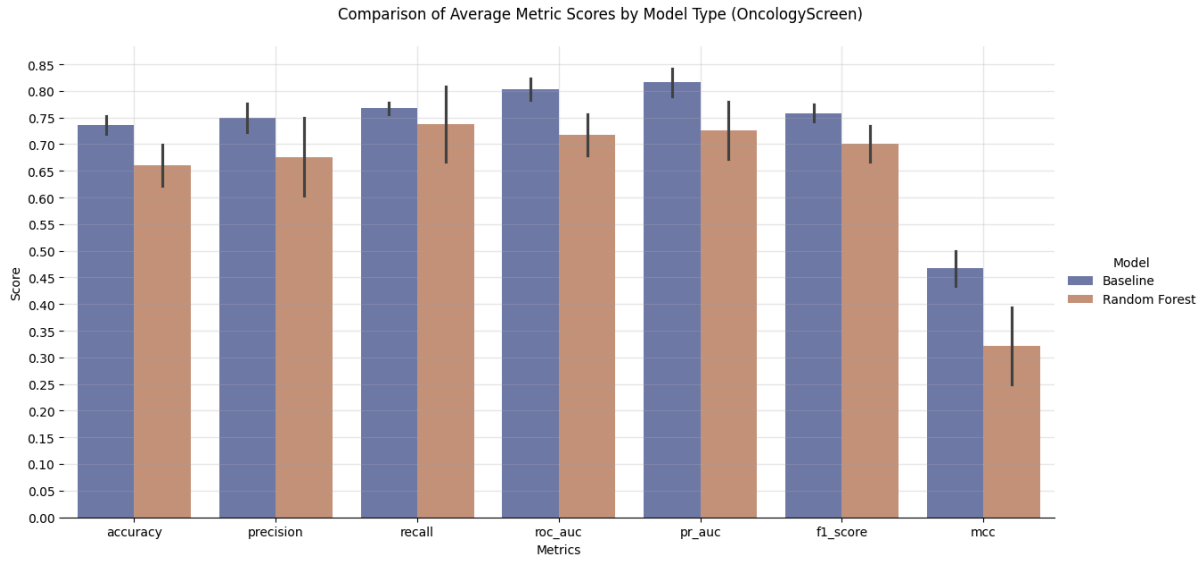
Fit of Trained Random Forest Model



Above are, side-by-side performance metric results on the train and test datasets. The Random Forest model seems to demonstrate good generalization capabilities, with both training and testing performance being on par, consistent and high across most metrics. There is again a higher volatility in the performance across folds in OncologyScreen, once again due to the low data volume. In either case there are no signs of overfitting or underfitting.

Random Forest vs GraphSynergy





When comparing the side-by-side performance results on the test datasets for GraphSynergy and Random Forest, we can see that Random Forest was slightly behind GraphSynergy in most metrics, while still achieving respectable accuracy with much better computational efficiency and better training and inference times. The low discrepancy between the results, suggest that a large portion of the performance was not dependent on the PPI-Graph.

One exception to the comparable performance can be seen in the large difference in the MCC metrics, with GraphSynergy outperforming Random Forest nearly 2 to 1 in that regard. While absolute performance was comparable, the

significant gap in MCC indicates that GraphSynergy captures more meaningful interactions, making it more reliable in real world applications.

Conclusion

In summary, we can say that we have not found any reasonable impact of the PPI graph at all. We were not able to reproduce what the paper claims. Additional explorations by us were that the synergy scores in the dataset are normally distributed, and the hyperparameters were likely to be tuned beforehand by the authors of the paper.

Our exploration of alternative models demonstrated that even very simple approaches can achieve high accuracy, highlighting the primary source of the model's performance. Additionally we demonstrated that our ComplexSynergy model can outperform the original model without using any information of the PPI Graph. While the PPI graph does contribute to the model's predictions, its overall impact is minimal.