Combination Antibiotic-Focused Machine Learning Models for Integration into Experimental Workflows

Combination Antibiotic-Focused Machine Learning Models for Integration into Experimental Workflows

A senior design project report submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering Sciences at Harvard University

Mythri Ambatipudi

S.B. Degree Candidate in Engineering Sciences

Faculty Advisor: Professor James Collins

Advisor: Jacqueline Valeri

Thesis Reader: Professor Sean Eddy

Harvard John A. Paulson School of Engineering and Applied Sciences and Engineering

Cambridge, MA

April 1, 2022

Table of Contents

List of Figures

List of Tables

Abstract

Antibiotic resistance is a growing crisis with far-reaching impacts on public and global health [1][2]. With antibiotic resistance impeding the efficacy of antibiotics and the discovery of new antibiotics slowing [2], combination therapies provide ways of extending the lifespan of existing antibiotics and treating resistant infections. Given the immense time and resources required to experimentally test combination therapies, computational strategies to accelerate the discovery of combination therapies are needed. In this project, machine learning models were developed to predict the synergistic activity of antibiotic-adjuvant pairwise combinations against *E. coli*. A variety of architectures were developed and tested. The most effective model type, a 2D Feed Forward Neural Network, was deployed through a Jupyter Notebook user interface for easy usage by experimentalists. Thus, this project engineers an accurate, computationally efficient, and user-friendly implementation of a machine learning algorithm that can integrate into experimental workflows to accelerate the discovery, screening, and validation of combination antibiotic therapies.

1 Introduction

1.1 Background and Motivation

Antibiotic resistance is an urgent public health issue [1][2]. Antibiotic resistance occurs when bacteria develop the ability to resist the growth-inhibition or lethality-promoting effects of the pharmaceutical agents used to treat infection by these bacteria. Consequently, infections of antibiotic resistant bacteria are significantly harder to treat, leading to complications among patients of these infections. Newly emerging resistant strains of bacteria such as *Clostridium difficile (C. difficile),* Carbapenem-resistant *Enterobacterales*, Vancomycin-resistant *Enterococcus*, and Methicillin-resistant *Staphylococcus Aureus* (MRSA) are concerning [3]. Over 2.8 million people in the U.S. are infected with antibiotic resistant strains annually and over 35,000 people die due to resistant infections [3]. The issue is magnified on a global scale, with over 700,000 people dying each year due to drug-resistant diseases [4]. Thus, antibiotic resistance is a prevalent issue with dire consequences.

Without newer, more potent antibiotics against which bacteria have not yet evolved resistance, treatment of infections will become increasingly challenging with existing antibiotics that are less effective against resistant bacteria. Therefore, since the utility of several existing antibiotics is dwindling due to antibiotic resistance, it is crucial to accelerate the discovery of new antibiotics against which resistance has not yet emerged. However, in the last few decades, antibiotic discovery has stalled. During the “Golden Age” of antibiotics in the 1960s, many easily-discoverable natural antibiotic products were identified [2]. Due to difficulties in identifying new bacteria that naturally produce antimicrobial agents [2], the rate of discovery of new antibiotics and classes of antibiotics has been significantly reduced. Shockingly, in the last four decades, almost no novel clinically relevant antibiotic classes have been discovered [1]. This stagnation in antibiotic discovery has perpetuated and aggravated the consequences of the antibiotic resistance crisis, with the time between antibiotic deployment and subsequent emergence of antibiotic resistance steadily decreasing [1].

Combination antibiotic therapies provide an avenue for alleviating current challenges in antibiotic discovery. Combination antibiotic therapies carry several advantages over single-agent antibiotics [5]. They are often used for broadening the antibacterial spectrum, i.e. granting the therapy the ability to target multiple bacterial species [5]. Especially when the cause of an infection in a patient is unknown, this broadening of the antibacterial spectrum increases the likelihood of at least one compound in the combination successfully combating the causative infectious agent. Along similar lines, combination antibiotic therapies are also used for polymicrobial infections, which are infections that are caused by multiple bacterial species [5]. Additionally, combination antibiotic therapies may cause beneficial synergistic antimicrobial effects that cause the potency of the combination to be greater than the potency of either agent alone [5]. They can also prevent or delay the detrimental effects of antibiotic resistance in patients [5], as synergy between compounds in a combination can re-sensitize bacteria to a therapy. While bacteria can rapidly develop resistance to single-agent antibiotics, it is less likely that bacteria will develop resistance to multiple agents, preserving the efficacy of the treatment for longer. Additionally, adjuvants can often re-sensitize bacteria to the antibiotic with which they are paired, as is the case when beta-lactam antibiotics are paired with beta-lactamase inhibitors [6]. These types of combinations are particularly advantageous, as the adjuvant chemical search space is significantly larger than the antibiotic chemical search space. However, as with all antibiotic therapies, if these combination therapies are used with improper specificity, this could lead to the emergence of more resistant strains of bacteria. This dilemma creates a heightened need for accurate prediction of effective combination antibiotic therapies.

It is therefore essential to rapidly accelerate the rate of accurate antibiotic discovery, especially for combination antibiotic therapies, in order to compensate for the shorter lifespan of antibiotics and the stagnation in new antibiotic discovery. Antibiotic screens to identify and test combination therapies are often time-consuming and painstaking for scientists. This is particularly aggravated by the rapid explosion in the number of pairwise combinations to test with the addition of more compounds. As an example, 100 compounds correspond to

4,950 pairwise combinations, which can be tested using 13 384-well plates. However, 1,000 compounds – a factor of 10 increase – corresponds to an astonishing 499,500 pairwise combinations and over 1,300 384-well plates for testing. Fortunately, recent advances in virtual high-throughput screens, computational techniques, and *in silico* drug design techniques have demonstrated immense potential for alleviating bottlenecks in the drug discovery pipeline [7][8][9]. Recent work by Stokes et al. [10] resulted in the utilization of one such computational technique that demonstrates the potential of artificial intelligence-inspired strategies to accelerate the drug discovery process. Stokes et al. employed a deep learning model for antibiotic discovery that was used to identify the broad-spectrum antibiotic efficacy of the molecule *halicin* [10]. Based on similar models created by Coley et al. in 2017 [11] and Yang et al. in 2019 [12], these models have also been used to predict other aspects of drugs more broadly, such as solubility and mechanism of action. Thus, artificial intelligence-based computational models that accelerate the discovery of combination antibiotic therapies may aid in alleviating the growing antibiotic crisis.

1.2 Problem Statement and Project Goals

In summary, antibiotic resistance poses a growing impediment to the efficacy of existing antibiotic therapies [2]. Combined with the stagnation in the rate of discovery of new antibiotics in the last few decades, this creates an urgent need for new tools that can rapidly increase the rate of discovery of antibiotic therapies capable of overcoming antibiotic resistance. Given the ability of combination antibiotic therapies to often demonstrate efficacy even against resistant bacteria, it is especially crucial to discover combination antibiotic therapies more rapidly. A computational artificial intelligence-based strategy for accelerating the discovery of combination antibiotic therapies can bridge existing gaps in the field.

Thus, the purpose of this project is to develop a machine learning model to predict the efficacy of combination therapies. The specific model proposed is an accurate neural network trained to predict whether a combination demonstrates synergistic antibacterial activity against *E. coli* using information derived from the molecular structure alone. Critical goals for this model are to achieve high accuracy, computational efficiency, and user-friendliness. The final product will provide a way to screen and validate combination therapies and gain insights for designing combinations more easily.

1.3 Potential Users and Stakeholders

The direct clients for my project are experimentalists in the Collins Lab at the Massachusetts Institute of Technology. The Collins Lab is interested in the discovery of combination antibiotic therapies, especially combinations where each individual agent is non-antibacterial but the combination as a whole is antibacterial. The Collins Lab faces numerous challenges in efficiently identifying new combination antibiotic therapies, including the fact that antibiotic screens to identify and test therapies are time-consuming and resource-intensive. The creation of a machine learning model can significantly aid in reducing this immense expenditure of time and effort by helping experimentalists better prioritize pairwise combinations to test and more rapidly identify the most likely adjuvant to synergize with a given antibiotic. Thus, the integration of this model into the workflow of experimentalists may accelerate the discovery of combination antibiotic therapies and will further the research efforts of the Collins Lab.

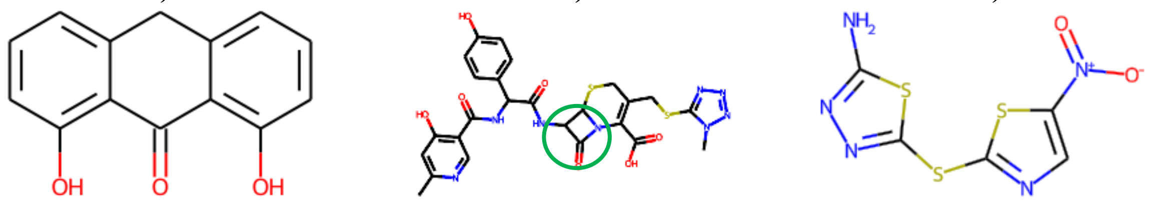
More broadly, in future, this model may advance efforts by scientists globally to develop combination antibiotic therapies and validate newly designed combination therapies. By providing a way to predict the antibacterial synergy of a combination, this model will offer a way for scientists designing combination antibiotic therapies to rapidly screen and validate antibiotic combinations. Interpretation of the mechanism of prediction of this model may grant scientists the ability to gain new insights for designing effective and potent antibiotic combination therapies. These benefits will in turn will aid in the rapid development of pharmaceutical treatments that clinicians can use to combat challenges posed by the growing antibiotic resistance crisis. By predicting the efficacies of multiple combinations at once, the model offers clinicians a list of substitute combination therapies that could potentially be used to treat patients exhibiting resistance to a particular single-agent or combination therapy. Thus, this model has the potential to benefit clinicians and patients in addition to scientists. Combination therapies are also used for treating other classes of diseases and infections, not just bacterial infections, for example as a hallmark of oncology treatments. If expanded for a broader scope, this model may additionally play a significant role in supporting scientists in their efforts to develop and identify combination therapies for a variety of medical fields.

2 Background Research

2.1 Machine Learning for Antibiotic Discovery

Recent efforts have demonstrated the ability of computational and artificial intelligence strategies to accelerate the discovery and development of antibiotics. In 2020, Stokes et al. used a graph neural network (GNN) deep learning model known as Chemprop [10]. This model, previously described in Coley et al. [11] and Yang et al. [13], was used by Stokes et al. directly for accelerating antibiotic discovery [10]. This model was applied to molecules from the Broad Institute Drug Repurposing Hub, a database containing 6,000 compounds that can potentially be used for novel therapeutic applications, and the ZINC15 database of over 1 billion compounds [10]. Application of the model to the ZINC15 database, in particular, led to the discovery of the bactericidal activity of halicin, especially against carbapenem-resistant Enterobacteriaceae and *Myocbacterium tuberculosis* [10]. This model, as well as the model that will be created in this project, makes these predictions using learned knowledge regarding molecular scaffolds and structures associated with antibiotics (Figure 1).

A) **B)** **C)**



Anthralin (non-antibiotic) Cefpiramide (antibiotic) Halicin (antibiotic)

Figure 1 - Molecular Structures Predicted by the Chemprop Model. A) The compound anthralin was classified by the Chemprop model as non-antibacterial due to its molecular structure [10]. **B)** The antibiotic cefpiramide, whose molecular structure and particularly the presence of the characteristic beta-lactam ring (circled in green) resulted in an antibacterial classification by the Chemprop model [10]. **C)** The novel antibiotic halicin was identified by the Chemprop model [10], demonstrating that this model is capable of not only predicting structures with traditional antibiotic scaffolds like beta-lactams but also generalizing to novel scaffolds and using the learned molecular structures to make predictions. (All of the above structures were generated using molecular visualization functions in the Python RDKit package)

Several other computational approaches such as molecular docking and virtual high-throughput screens have been used for antibiotic discovery. Many machine learning models have been used for antibiotic discovery and for identifying mechanisms of action for antibiotics. For example, machine learning models have been created for applications such as target validation, biomarker identification, and clinical trial data analysis [14]. In the Collins Lab, a white box machine learning approach has been created to better understand the causal mechanisms of action of antibiotics [12]. Application of this model revealed that biochemical pathways related to purine biosynthesis are involved in the mechanisms of action contributing to antibiotic lethality [12]. This model provided a novel way of identifying the mechanistic basis for antibiotic activity, but it relies on metabolic data from experimental screens [12], increasing the necessary investment of time and experimental effort. Another model, Mol2vec, is an unsupervised machine learning approach for converting molecular structures to vectors [15]. This model was used for compound property and bioactivity prediction with an auROC of 0.86 when predicting the aqueous solubility of compounds, 0.83 when predicting the toxicity of compounds, and 0.87 when predicting the mutagenicity of compounds [15]. However, it was not directly tested with antibiotic prediction [15]. The vast field of machine learning applied to molecular drug discovery is outside of the scope of this report, but advances summarized by Elton et al. [16], Romano et al. [8], and Ekins et al. [17] yield great promise for *in silico* predictions. Thus, several existing machine learning approaches, like the aforementioned white box machine learning approach and Mol2vec, demonstrate immense potential for predicting molecular properties and behaviors. However, the limitations of these models, such as reliance on data from time-consuming experimental screens, may also impede their ability to be effectively and efficiently used for rapid antibiotic discovery.

In contrast, several GNNs have demonstrated enormous success with both efficient and accurate antibiotic discovery and prediction of antibiotic properties. GNNs are especially useful for molecular machine learning because the message-passing structure of GNNs allows for the propagation of information within a local neighborhood of the molecule. GNNs represent atoms and bonds in compounds as networks of nodes and edges. This representation may be advantageous in capturing more structural features than Morgan fingerprints, which are vector representations of molecular structures in which each substructure in the molecule is hashed to a bit, and due to the limited number of bits in the fingerprint, multiple substructures can sometimes be hashed to the same bit, a phenomenon known as collision [18]. In 2017, Coley et al. created a GNN for predicting the physical properties of drug molecules, such as aqueous and octanol solubility, melting point, and toxicity [11]. This model provided the foundation for the previously described model created by Yang et al. for property prediction [13] and later used by Stokes et al. [10]. Another GNN created by Coley et al. in 2019 is capable of predicting chemical reactivity [19]. This GNN, trained on reaction precedents spanning numerous reaction types, predicted the major product of a reaction with high accuracy (85%) and speed (100 milliseconds per example) [19]. The unique structure of GNNs, specifically the alignment of its approach with that of chemists, was thought to contribute to this high performance [19]. Thus, GNNs provide a unique approach to molecular modeling that makes them a particularly advantageous choice for antibiotic-targeted computational modeling.

2.2 Machine Learning for Drug Combination Discovery

In addition to these single-agent antibiotic discovery approaches, several novel computational efforts for combination drug discovery have also made significant advances. However, many of these approaches require the investment of additional time and effort by the user, thereby limiting their efficacy. For instance, a computational approach called INDIGO makes predictions regarding antibiotic combinations that act synergistically and antagonistically [20]. INDIGO was trained using experimental drug interaction data as well as chemogenomic profiles [20]. Another similar model, MAGENTA, predicts the efficacy of drug combinations in different pathogen microenvironments [21]. MAGENTA uses a Random Forest model to predict synergistic or antagonistic interactions between drugs in different environments, such as in rich LB growth media and minimal glucose growth media [21]. However, both these models require chemogenomic profiling data, which is often limited or inaccessible. The collection of such data requires additional experimental testing, lessening the efficiency of the approach. Wood et al. identified scaling laws, which are ways of quantitatively describing the functional relationship between two variables, and used them to predict the responses of drug-sensitive and drug-resistant bacteria to multidrug combinations [22]. However, the disadvantage of this approach is that it requires dose data, i.e. data demonstrating the response of bacteria to different doses/concentrations of molecular agents. Like the chemogenomic profiling data required for INDIGO and MAGENTA, the collection of dose data for Wood et al.'s approach requires experimental screens that require additional time and effort, thereby reducing the efficiency of the approach. In addition to these models that predict efficacy, other models focus on dosages of combination therapies. For example, Smith et al. implemented machine learning algorithms for identifying the optimal dosing strategies for the combination of meropenem and polymyxin B against carbapenem-resistant *Acinetobacter baumannii* [23]. This model is a particularly significant example of the utility of machine learning in identifying combination therapies that remain effective against drug-resistant strains. However, this model is applicable to only a specific antibiotic combination and also requires experimental data as input. Once again, this requirement for experimental data increases the investment of time and effort, thereby reducing the efficiency of the approach.

Finally, while numerous models for combination antibiotic prediction exist, several models have also been created for drug combinations outside the antibiotic space. For instance, Li et al. created an ensemble machine learning model to predict drug combinations using similarity-based multifeatured drug data [24], and Kim et al. created a deep neural network for predicting drug combinations that maximize anti-aging effects [25]. These models demonstrated strong performance, but both require the collection of data beyond just the molecular structures, such as the differential gene expression data required by the Kim et al. model [25]. Along similar lines, Julkunen et al. created comboFM, a machine learning model that predicts the effects of drug combinations in preclinical experiments [26], and Liu et al. created TranSynergy, an interpretable deep learning interpretable model for predicting synergistic anti-cancer drug combinations [27]. These models have significant advantages, including the excellent interpretability of the TranSynergy model, but both once again require additional experimental data. TranSynergy, in particular, requires drug-target data [27], which can pose a significant limitation when exploring a set of molecules that are not yet approved drugs and therefore have unknown targets. Finally, Jin et al. recently created a model for discovering synergistic drug combinations for COVID-19 [28]. This model learns both information regarding drug-target interactions and synergy, which grants the model the ability to predict how the synergy between drugs in a combination change based on the biological target [28]. It uses a GNN architecture for predicting drug-target interactions and architecture based on Bliss synergy scores for predicting synergy [28]. This model was shown to have strong performance (auROC of 0.78) [28], further demonstrating the potential of GNNs to accelerate the discovery of drug combinations for COVID-19.

2.3 Prior Work

Previously, as part of my work for my ES91r project, which was also conducted as part of the Collins Lab, random forest classifiers, support vector machine classifiers, and K-nearest neighbors classifiers were created for single-agent and combination therapy predictions. The random forest was created using the Scikit-learn RandomForestClassifier package in Python. Different numbers of trees (500, 1000, 2000, 3000, and 5000 trees) were tested to create the most accurate model. Similarly, the SVM classifier was created using Scikit-learn svm package in Python, and the KNN classifier was created using the Scikit-learn neighbors package in Python. The neural network models were created with TensorFlow and the Chemprop GitHub repository.

For single-agent prediction, data from Stokes et al. [10] was used to train and test the models. Stokes et al. performed a large, *in vivo* screen of 2,335 molecules to determine their antibacterial activity against *E. coli*. SMILES (Simplified Molecular-Input Line-Entry System) string representations of each molecule's structure were converted into Morgan Fingerprints using the Python RDKit package [29]. The highest performing model achieved an auROC of 0.83, which falls below the auROC of 0.896 achieved by the GNN created by Stokes et al. [10]

For combination therapy prediction, the models were trained and tested using data from Kulesa et al. [30]. The original dataset provides SMILES string representations of various molecules, including 10 antibiotic molecules and 2,751 non-antibiotic adjuvant compounds. First, the SMILES representation of the full combination was created by appending the SMILES representation of the antibiotic, followed by a period, and then the SMILES representation of the adjuvant compound. The period signals to RDKit that the combination should be treated as two disconnected halves of a single drug. These SMILES strings were then converted to 2,048-bit Morgan Fingerprints using the Python RDKit package [29]. For each combination of 1 of the 10 antibiotics with 1 of the 2,751 other compounds, a Bliss synergy score and p value is provided. The Bliss synergy score is a method of quantifying the synergy between the two molecules in the combination [31]. It can be calculated by calculating the difference between the observed growth inhibition induced by a combination and the predicted inhibition rate from the Bliss independence equation, which is:

$$\hat{y}_{ab} = y_a + y_b - y_a y_b \tag{1}$$

where y_a and y_b are the inhibition rates induced by drug A and drug B respectively at dosages of a and b and $\hat{y}_{ab}$ is the predicted inhibition rate assuming Bliss inhibition [31]. The difference between the predicted inhibition rate $\hat{y}_{ab}$ assuming Bliss independence and the actual observed inhibition rate was used by Kulesa et al. to determine the final Bliss synergy score [30]. In this final Bliss synergy score, greater values indicate stronger synergy, while lower values indicate weaker or absent synergy. In addition to this Bliss score, the p values provided in the dataset indicate the significance of the interaction between molecules. These two metrics were converted into a binary score indicating whether each combination demonstrates antibacterial synergy. To be classified as a "hit" demonstrating synergy, a p value threshold of 0.05 and a Bliss synergy score threshold of 0.5 were used, i.e. combinations with a p value < 0.05 and a Bliss synergy score > 0.5 were assigned a value of 1 (indicating that they are hits) while combinations with a p value ≥ 0.05 and a Bliss synergy score ≤ 0.5 were assigned a value of 0 (indicating that they are non-hits). Using this classification system, ~0.63% of the total number of combinations in the dataset were classified as "hits," meaning that 0.63% of the antibiotic-adjuvant combinations demonstrate synergy. Thus, the final Morgan Fingerprint molecular representations of the entire combination structures, along with these binary synergy scores, were used to train and test the models.

For combination therapy classification, the auROCs for the random forest classifier reached values no higher than 0.5 (no better than a random classifier), and the auROCs for the neural networks attained a maximum value of ~0.75. These models therefore did not perform with accuracy, though not much fine-tuning of model parameters or input parameters was performed. In this ES100 project, more advanced models were created, new ways of providing input features to these models were explored, and unique modifications to the model designs were made in order to achieve accuracy and model utility. The models created in this ES100 project were trained and tested using the processed version of the Kulesa et al. [30] dataset. Thus, the prior work was a limited baseline exploration of the work which was conducted in this ES100 project.

2.4 Remaining Unmet Needs

In light of this prior work, a need remains for a computationally efficient, accurate model for accelerating the discovery of combination antibiotic therapies. Such as the Chemprop model applied to single-agent predictions (Figure 1), it is especially important for such a model to learn characteristics of molecular scaffolds (both traditionally known and novel) that aid in more accurately and efficiently classifying combinations as antibiotic or non-antibiotic. As discussed, several models have been created for predicting the efficacy of single-agent antibiotic therapies, the efficacy of and bacterial response to combination antibiotic therapies, and optimal dosages of antibiotic combination therapies. While some models have been created for combination antibiotic therapies, they are often computationally inefficient and require additional data (such as chemogenomic profile or dose data), which further limits the ease of use and efficiency of the overall approach. Thus, a need exists for computational models that can integrate easily into experimental workflows to conserve time and improve efficiency for experimentalists. To manually test such large numbers of pairwise combinations (often on the order of magnitude of 10K x 10K) is often unfeasible for experimentalists. Therefore, it is important to create such computational models that can integrate into their workflows and reduce time and effort expended.

Thus, this project will focus on creating an accurate, computationally efficient, and user-friendly model that can predict the efficacy of combination antibiotic therapies using just their molecular structures as minimal information.

3 Design Goals and Technical Specifications

3.1 Overarching Design Goals

There are three broad goals in the design of the final product for this project: 1) high accuracy, 2) low computational intensiveness, and 3) user-friendliness.

Each goal has far-reaching impacts on various non-technical contexts, including public and global health and safety (Appendix 2). Successful combination antibiotic therapies are crucial for effectively treating bacterial infections, and with the growing public and global health threat posed by antibiotic resistance, it is essential to accelerate the discovery of effective combination antibiotic therapies. It is therefore important for the model developed in this project to 1) provide accurate predictions that experimentalists can reasonably rely on to accelerate the testing, validation, and deployment of antibiotic therapies and 2) be delivered to experimentalists in a platform that they can easily use to obtain the information that is most useful to them (Appendix 2). Thus, high accuracy, low computational intensiveness, and user-friendliness are all crucial to meeting these broader requirements.

Similarly, these three main goals also have economic and environmental implications (Appendix 2). Currently, experimental screens of combination antibiotic therapies have very low hit rates, so a drastically large percentage of the tested combinations are not successful synergistic combinations. Consequently, these large-scale assays result in an enormous expenditure of resources and materials, resulting in economic and environmental losses. In order to alleviate these losses, reduce the number of combinations to be tested, and increase the success rate, it is important for the model to have high accuracy (Appendix 2).

3.2 Quantitative Technical Specifications

There are a number of overall requirements and specifications (both quantitative and qualitative) that guide the design of the final product in this project. Table 1 below shows a summary of these specifications, and additional discussion of each specification follows below.

Table 1. Summary of technical specifications

Specification	Value	Measurement	Existing Solutions	Justification
Model Accuracy				
False Positive Rate (FPR)	≤ 0.1	Python Scikit-learn metrics package	Chemprop FPR for single-agent predictions = 0.485 [10].	- False positives result in enormous waste of experimental time and resources. - Should perform better than existing solutions.
False Negative Rate (FNR)	≤ 0.5	Python Scikit-learn metrics package	Chemprop FNR for single-agent predictions = 0.032 [10].	- False negatives result in premature elimination of potentially successful combinations. - While keeping the FPR low, it is often infeasible to achieve equally low FNR [32][33].

auROC	≥ 0.8	Python Scikit-learn metrics package	Chemprop auROCs for single-agent predictions ranged from ~0.6-0.9 for different applications [10] [13].	- Should be accurate in predicting effective combinations to be useful to experimentalists. - Should be at par with the best-performing existing single-agent models, but combination predictions are more difficult.
auPR	≥ 0.2	Python Scikit-learn metrics package	Chemprop auPRs for single-agent predictions ranged from ~0.05-0.4 for different applications [10] [13].	- Lower thresholds are appropriate when using datasets with a low percentage of positives. - Should be at par with the best-performing existing single-agent models, but combination predictions are more difficult.
Computational Intensiveness				
Model Speed	≥ 1200 combos/s when run on CPU with at least 16 GB RAM, 256 GB storage	Measured elapsed time for model to finish running with a set # of combinations	Chemprop GNN [13] had processing speed of ~1500 single-agent molecules per sec.	- Should be fast but willing to sacrifice some speed for model and combination complexity and accuracy. - Should be roughly comparable to existing solutions. - Time to process batch of combinations should be within range of experimentalists' expectation.
RAM required to run model	≤ 8 GB	Activity Manager/Task Manager	Chemprop GNN [13] runs on standard laptops with 8 GB RAM.	- Usually the minimum laptop RAM that can successfully runs machine learning models [34]. - Should be comparable to existing solutions.
Storage required to download model	≤ 2.5 GB	Size of folder containing model and all necessary dependencies	Entire Chemprop [13] repository and associated dependencies occupy ~2.3 GB of storage.	- Should not occupy excessive computer storage. - Should be comparable to similar packages, but some leeway to allow for model complexity and accuracy.
Largest input dataset file size allowed by the model	≥ 1 GB	Test different file sizes to see which will overload the model	Most models use Pandas, which functions well for up to 1 GB file sizes [35].	- Should be able to use datasets of at least the maximum file size that the Python Pandas library can successfully process.
User-Friendliness				
Survey results from experimentalists	Report ease/comfort of integrating into workflow Rating of ≥ 4 out of 5	Provide Collins Lab experimentalists with rating scale and calculate average rating	Other models have not been assessed using this scale.	- Should be user-friendly but clients have experience in using similar tools.

3.2.1 Accuracy

The first set of technical specifications pertain to the accuracy of the predictions made by the machine learning model.

3.2.1.1 False Positive Rate (FPR)

Quantitative Description

The false positive rate (FPR) should be no greater than 0.1, i.e. FPR ≤ 0.1. This FPR specification applies for running the model on training and testing datasets.

Justification

False positives carry high costs, as a high false positive rate implies enormous waste of experimental time, effort, and resources. Given that all antibiotic combinations that are predicted to demonstrate synergistic antibacterial activity by the model created in this project must be validated experimentally, minimizing the unnecessary expenditure of experimental time and resources is crucial. Experimentalists in the Collins Lab, the clients for this project, frequently experience antibacterial screen success rates of ~10%, where only 10% of the compounds they screen succeed. For this reason, it is important to minimize the FPR of the model created in this project, as this will aid the model in making a significant contribution to increasing these experimental screen success rates without further compounding the issue of high false positives rates. Currently, many computational efforts for antibiotic prediction have resulted in moderate to high FPR values. For instance, Stokes et al. observed a true positive rate (TPR) of 0.515 (FPR of 0.485) when testing the Chemprop GNN for single-agent antibiotic discovery [10]. Therefore, to achieve improved accuracy compared to existing models like Chemprop and aid in increasing the success rates of experimentalists during antibiotic screens, an FPR threshold of 0.1 was used.

Measurement

To measure the FPR, the calculation shown in Equation 2 below is used:

$$FPR = \frac{FP}{(FP + TN)} \tag{2}$$

where FP is the number of false positives and TN is the number of true negatives. These calculations can be performed in Python using built-in functions in the Scikit-learn package [36].

3.2.1.2 False Negative Rate (FNR)

Quantitative Description

The false negative rate (FNR) should be no greater than 0.5, i.e. FNR ≤ 0.5. This FNR specification applies for running the model on training and testing datasets.

Justification

If a combination with antibiotic efficacy is predicted to not have antibiotic efficacy, this would result in the premature elimination of a combination therapy that may demonstrate utility. Consequently, it is important to maintain a relatively low FNR. However, in similar molecular models, maintaining an extremely low FNR while simultaneously maintaining a low FPR is often very difficult or infeasible [32][33]. For instance, in a computational virtual screen of antibacterial drugs, Durrant et al. found that a FPR of 5% corresponded to a

TPR of 47% (FPR of 53%) [32]. Furthermore, when developing DeepARG, a deep learning approach to predict antibiotic resistance genes, Arango-Argoty et al. observed that low false positive rates occurred in conjunction with high false negative rates [33]. Another important factor is the greater complexity involved in combination antibiotic therapy predictions compared to single-agent therapy predictions, as the molecular structures of two compounds as well as interactions between them must be considered by a combination-predicting model. In this project, the priority is to maintain a high FPR in order to ensure that the created model has a significant utility to experimentalists, as described previously in Section 3.2.1.1. Therefore, keeping in mind the tradeoff between FPR and FNR as well as the increased difficulty in combination antibiotic predictions, a FNR threshold of 0.5 was used.

Measurement

To measure the FNR, the calculation shown in Equation 3 below is used:

$$FNR = \frac{FN}{(FN + TP)} \tag{3}$$

where FP is the number of false positives and TN is the number of true negatives. These calculations can be performed in Python using built-in functions in the Scikit-learn package [36].

3.2.1.3 Area Under Receiver Operating Characteristic Curve (auROC)

Quantitative Description

The auROC should be at least 0.8, i.e. auROC ≥ 0.8. This auROC specification applies for running the model on training and testing datasets.

Justification

The baseline auROC for a random classifier is 0.5. Thus, for a non-random, high-performing classifier, the auROC should be much higher. Prior neural network models created by Yang et al. and Stokes et al. for a variety of molecular prediction applications (e.g. predicting toxicity, efficacy, etc.) achieved auROC values ranging from ~0.7-0.9 [13][10]. However, it is important to keep in mind that many of these models are for predictions regarding single-agent molecules. In contrast, predictions regarding combinations of molecules are more complex and difficult, so achieving the highest auROC values is less feasible. The aim for this project was to achieve auROC values that are at least at par with the moderately well-performing existing single-agent models. Based on this existing literature [13][10], an auROC threshold of 0.8 was therefore used.

Measurement

To measure the auROC, the TPR is plotted against FPR, and the area under the resulting curve is measured. These calculations can be performed in Python using built-in functions in the Scikit-learn package [36].

3.2.1.4 Area Under Precision Recall Curve (auPR)

Quantitative Description

The auPR should be at least 0.2, i.e. auPR ≥ 0.2. This auPR specification applies for running the model on training and testing datasets.

Justification

The baseline auPR for a random classifier depends on the percentage of positive "hits" in the dataset. The dataset used in this project has imbalanced classes, which means the proportion of negatives in the dataset is significantly greater than the proportion of positives. Since there is therefore a small percentage of positives, the baseline auPR is low. auPR values seen by Yang et al. for molecular models for several applications (e.g. molecular toxicity, drug side effects, etc.) were variable, but average auPR values of ~0.1-0.2 were observed [13]. Thus, based on values seen in the literature for such models using datasets that likewise have small percentages of positives [13], 0.2 is an adequate threshold for minimum auPR in this project.

Measurement

To measure the auPR, model precision is plotted against model recall, and the area under the resulting curve is obtained. Precision and recall are calculated as shown in Equations 4 and 5 below:

$$Precision = \frac{TP}{(TP + FP)} \tag{4}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{5}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives. These calculations, as well as the calculation of the area under the curve when precision is plotted against recall, can be performed in Python using built-in functions in the Scikit-learn package [36].

3.2.2 Computational Intensiveness

The second group of technical specifications pertain to the computational intensiveness of the machine learning model.

3.2.2.1 Model Speed

Quantitative Description

The model should be able to process combinations with a speed of ≥ 1,200 combinations per second on the standard hardware used by Collins Lab experimentalists (typically Mac CPU with 16 GB RAM, 256 GB storage).

Justification

The model speed metric was selected based on values seen in literature. The Chemprop GNN used by Stokes et al. [10] demonstrates a processing speed of 1,500 molecules/second. To achieve high accuracy, greater computational intensiveness is sometimes required. To allow some leeway for more complex combinations representations than the Chemprop model, a slightly lower minimum processing speed of 1,200 combinations/second should be used.

Additionally, the clients of this project, Collins Lab experimentalists, typically aim to process 1,000-10,000 combinations in one batch and expect the entire workflow to take 24 hours with the aid of a liquid-handling robot. Since there are several preparation, validation, and interpretation steps that also consume time, the process of running this model should require no more than a few seconds for a batch of combinations. Using a model speed of 1,200 combinations/second, 10,000 combinations can be processed in ~ 8 seconds. Thus, a model speed of at least 1,200 combinations/second would aid in ensuring that the time devoted to running the model in the experimental workflow is within range of the experimentalists' expectations.

Measurement

To measure the model speed, the number of combinations input into the model and the elapsed time for the model to finish running are recorded. Specifically, the model speed technical specification should be met when the model is run on computer CPUs that have hardware specifications greater than or equal to those typically used by Collins Lab experimentalists: 16 GB RAM and 256 GB storage.

The model speed (combinations/second) is then calculated using the relationship in Equation 6 below:

$$Model\ Speed = \frac{\#\ of\ input\ combinations}{Time\ for\ model\ to\ run} \quad (6)$$

3.2.2.2 RAM Required to Run Model

Quantitative Description

The model should require ≤ 8 GB of RAM to run successfully.

Justification

Although most of the Collins Lab experimentalists use CPUs with 16 GB of RAM, the model should be able to run on any computer that has the capability to run machine learning models to make the model as widely accessible as possible. The model should therefore have a maximum RAM usage of 8 GB, as that is usually the minimum laptop RAM that can successfully runs machine learning models [34].

Measurement

To perform the measurement of the RAM usage, built-in functions in the Python psutil package [37] can be used. These functions allow for the measurement of RAM usage at different points in the code.

3.2.2.3 Storage Required to Download Model

Quantitative Description

The model and all its associated dependencies should together occupy ≤ 2.5 GB of computer storage.

Justification

A maximum storage requirement of 2.5 GB was chosen based on the storage requirement for the Chemprop model [13]. The entire Chemprop repository and associated dependencies occupies ~2.3 GB of storage. Once again, to allow some leeway for greater accuracy and complexity than the Chemprop model, a slightly higher maximum storage requirement of 2.5 GB should be used. It is also important that the model does not consume an excessively large portion of the computer storage. Thus, out of the 256 GB storage available in the computers used by most Collins Lab experimentalists, this tool should not occupy any more than 2.5 GB, or roughly 1%, of storage.

Measurement

To measure the required storage, the storage space (GB) occupied by the entire model repository and its necessary dependencies is assessed. This is measured by assessing the size of the folder that contains all model-associated files.

3.2.2.4 Largest Input File Size Allowed by Model

<u>Quantitative Description</u>

The size of the largest input file that can be loaded and processed by the model should be at least 1 GB.

<u>Justification</u>

The input dataset file size is an important specification, as it determines how many combinations can be processed by the model simultaneously. The model should be able to load and process files that are at least as large as the maximum file size that can be used by other Python data analysis libraries, i.e. the model should not add an additional constraint. Thus, this specification was set based on the ability of Pandas, a Python data analysis library, to load and process large files. Pandas functions fairly well for files up to ~1 GB in size, but with larger files, some modifications are needed to achieve normal performance [35]. For this reason, the final product created in this project should be able to load and use datasets at least as large as this 1 GB threshold.

<u>Measurement</u>

To measure the largest input file size allowed, the largest file size of the dataset of compounds/combinations that can be input into the model without overloading the model is recorded. This is largely done through manually testing different file sizes to determine which file size will hinder the model to an extent that prevents it from running.

3.3 Qualitative Technical Specifications

In addition to these quantitative technical specifications, there is also a final qualitative technical specification guiding the final product in this project.

3.3.1 User-Friendliness

The key qualitative requirement for the final product is that it should be feasible for Collins Lab experimentalists to integrate the usage of this model into their workflows without additional training.

3.3.1.1 User Survey Results

<u>Description</u>

The clients of this project, the scientists in the Collins Lab, should report utility of the model and ease in integrating it into their workflows. The experimentalists should also report being able to comfortably use the tool without additional training. As an objective basis for evaluation, the final tool should also achieve an average user rating of at least 4 out of 5.

<u>Justification</u>

The goal is for this model to integrate easily into the workflow of experimentalists. Thus, the end product should be user-friendly, i.e. users should report ease and comfort in integrating it into their experimental workflows, and it should have a high rating score out of 5. Experimentalists in the Collins Lab have had specific training in using Jupyter Notebooks through the MIT Biological Engineering department's core classes 20.420 and 20.440. Experimentalists should thus be able to utilize the final product easily and comfortable without requiring additional training or expertise.

Measurement

To measure user-friendliness, the intended clients are provided the final product and a survey to evaluate their experience when using the product. While the specific questions asked in this survey are specific to the goals of Collins Lab experimentalists, existing user-friendliness surveys, such as the System Usability Scale [38], guide the questions asked in this survey (e.g. questions regarding user's perception of tool complexity, user's confidence while using tool, user's belief regarding whether they would need external support to use the tool, etc.). Qualitative feedback is collected from survey respondents to identify weaknesses and improve the user-friendliness of the final product, and a rating of the final product on a scale of 1-5 is also collected.

4 Technical Design Approach

4.1 Overview of Key Design Components

There are a few main components for the design of the final product being created in this project: a ground truth dataset to train, test, and refine the machine learning model; the machine learning model itself; and a user interface. Figure 2 below shows a high-level flowchart demonstrating the relationships between these three main groups of components.

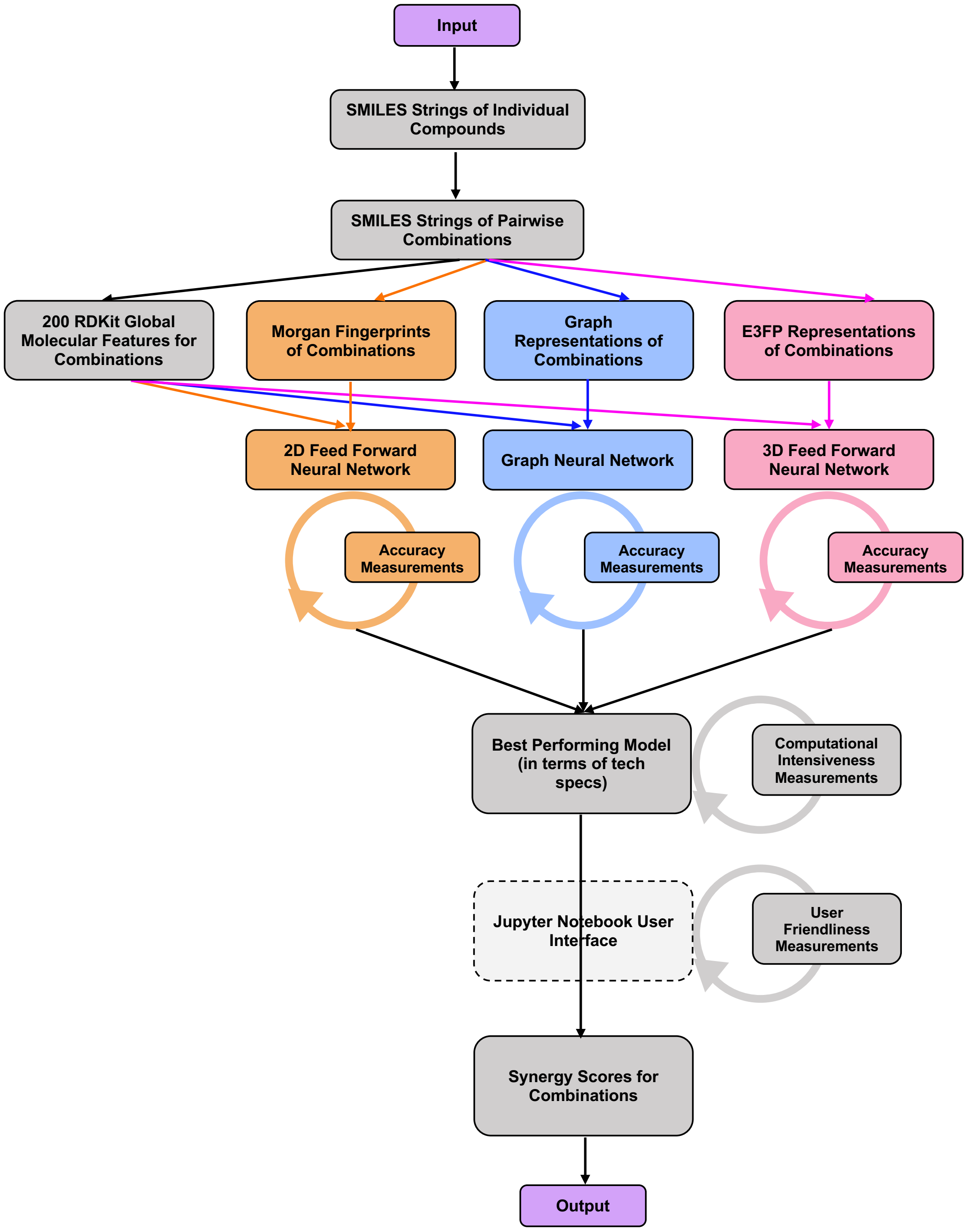

Input
SMILES Strings of Individual Compounds
SMILES Strings of Pairwise Combinations
200 RDKit Global Molecular Features for Combinations
Morgan Fingerprints of Combinations
Graph Representations of Combinations
E3FP Representations of Combinations
2D Feed Forward Neural Network
Graph Neural Network
3D Feed Forward Neural Network
Accuracy Measurements
Accuracy Measurements
Accuracy Measurements
Best Performing Model (in terms of tech specs)
Computational Intensiveness Measurements
Jupyter Notebook User Interface
User Friendliness Measurements
Synergy Scores for Combinations
Output

Figure 2 - Design Diagram. The flowchart demonstrates how the different components of the project fit together. Purple represents what the user sees (only the inputs and the outputs). Green represents the steps/components independent of the model class (data preparation, Jupyter Notebook interface creation, etc). Orange represents the steps/components specific to 2D FFN model design. Blue represents the steps/components specific to GNN design. Pink represents the step/components specific to 3D FFN design. Note that the 2D FFN, GNN, and 3D FFN will be developed sequentially.

4.1.1 Ground Truth Dataset

A ground truth dataset is a critical component of the complete design. To evaluate model performances at each stage of the project, whether during the process of designing model architectures, while training or testing models, or during the final evaluation of models, a ground truth dataset is essential to provide known true values against which to compare model predictions.

In this project, data from Kulesa et al. [30], pre-processed as previously described [see Section 2.3], serves as this ground truth dataset. Each model developed in this project is trained and tested, both while identifying the optimal design and while testing the final models, using the data from this ground truth dataset. Information regarding the molecules contained within this dataset serve as the inputs to each model. Given that different model types often require different types of input data, information regarding the molecules in the ground truth dataset is represented in different ways prior to being input into each model. The binary synergy scores (1 = synergistic, 0 = non-synergistic), generated from Bliss synergy scores and p values contained in this dataset as previously described, serve as the ground truth output, which each model must learn to predict from the given inputs.

4.1.2 Machine Learning Model

The machine learning model comprises the bulk of the overall design of the product being created in this project. The model is the component of the overall design responsible for generating the predictions that are intended to aid experimentalists in their testing of combination antibiotic therapies.

The model requires a list of SMILES string representations of compounds as input from the user. SMILES (Simplified Molecular Input Line Entry Specifications) strings [39] are a text-based representation of the individual atoms and bonds in each molecule (Figure 3). The advantages of SMILES strings are that they are more compact than other molecular representation methods and are easier to interpret and manipulate by a user [39]. These SMILES string representations of individual compounds are then used to create SMILES string representations of combinations by appending them with a period in between the two molecules, which represents the two compounds in the combination as two disconnected halves of a single drug. All information needed to serve as input data for the model is then extracted using these SMILES string representations of the combinations.

This model design is that of a classifier, so it is intended to help the user decide if a combination therapy demonstrates synergistic antibacterial activity or not. Thus, the output for this model design is the model's prediction regarding the probability of a given combination being synergistic. Specifically, for each combination, the model outputs a value that represents the probability of the combination demonstrating synergy. Other classifiers sometimes output binary values that are either 0 or 1. However, outputting a value that is indicative of probability is more informative, as it provides the user an assessment of the model's confidence in its classification.

Three different classes of model designs are tested in this project: 2D Feed Forward Neural Network (2D FFN), Graph Neural Network (GNN), and 3D Feed Forward Neural Network (3D FFN). The type of model class impacts various aspects of the final model performance, especially the model accuracy and the computational intensiveness. Each of these three model classes has its own set of advantages and disadvantages, discussed further in detail below.

4.1.3 User Interface

The final user interface is a Jupyter Notebook that allows experimentalists to use the final machine learning model to make predictions regarding the synergistic activity of combination therapies in their own datasets. A user interface that best suits the expertise and needs of the experimentalists is essential for ensuring that the clients of this project are able to utilize the machine learning model in a user-friendly and efficient manner, with minimal inconvenience or uncertainty. The importance of user-friendliness in meeting the broad goals of this project and addressing several non-technical contexts (Appendix 2) was previously described. The user interface is the component of the design that most directly addresses these goals and requirements.

4.2 Ground Truth Dataset

As previously described, two key sets of information generated from the dataset from Kulesa et al. [30], the SMILES strings representations of combination therapies and binary synergy scores, comprise the ground truth dataset in this project. SMILES strings provide information regarding molecule structure, but they are not always unique, so different molecules may have the same SMILES string representation [16]. Additionally, there is a very limited number of model architectures that can make predictions using text-based representations as inputs. Consequently, molecular representations that can more readily be vectorized, generated from the SMILES string representations, serve as the inputs for each type of model.

For the 2D FFN, the appropriate molecular representation method is Morgan Fingerprints. Morgan Fingerprints are vector representations of molecular structures (Figure 3). Each bit in the Morgan Fingerprint represents one or more substructures that are hashed with collision to the same bit [see Section 2.1] (Figure 3). Morgan Fingerprints are more difficult to directly interpret by a user compared to SMILES strings, but they are particularly advantageous for representing substructures around each atom in a molecule and for making predictions related to small molecules [40].

For the GNN, the appropriate molecular representation method is molecular graphs. Molecular graphs represent the atoms and bonds in compounds as networks of nodes and edges (Figure 3) and may be used to capture more complex structural features than models trained on Morgan fingerprints. These advantages of molecular graphs make them a popular choice for making broad chemical predictions [19][11][10].

For the 3D FFN, the appropriate molecular representation method is E3FP fingerprints. E3FP representations are spherical extended 3-dimensional molecular fingerprints [41] (Figure 3). While Morgan Fingerprints are 2-dimensional fingerprints, E3FP fingerprints are similar vector representations of molecular substructures in three dimensions [41] (Figure 3). Since the E3FP fingerprints provide information in the third dimension, more information is made available to the machine learning model, potentially improving model performance.

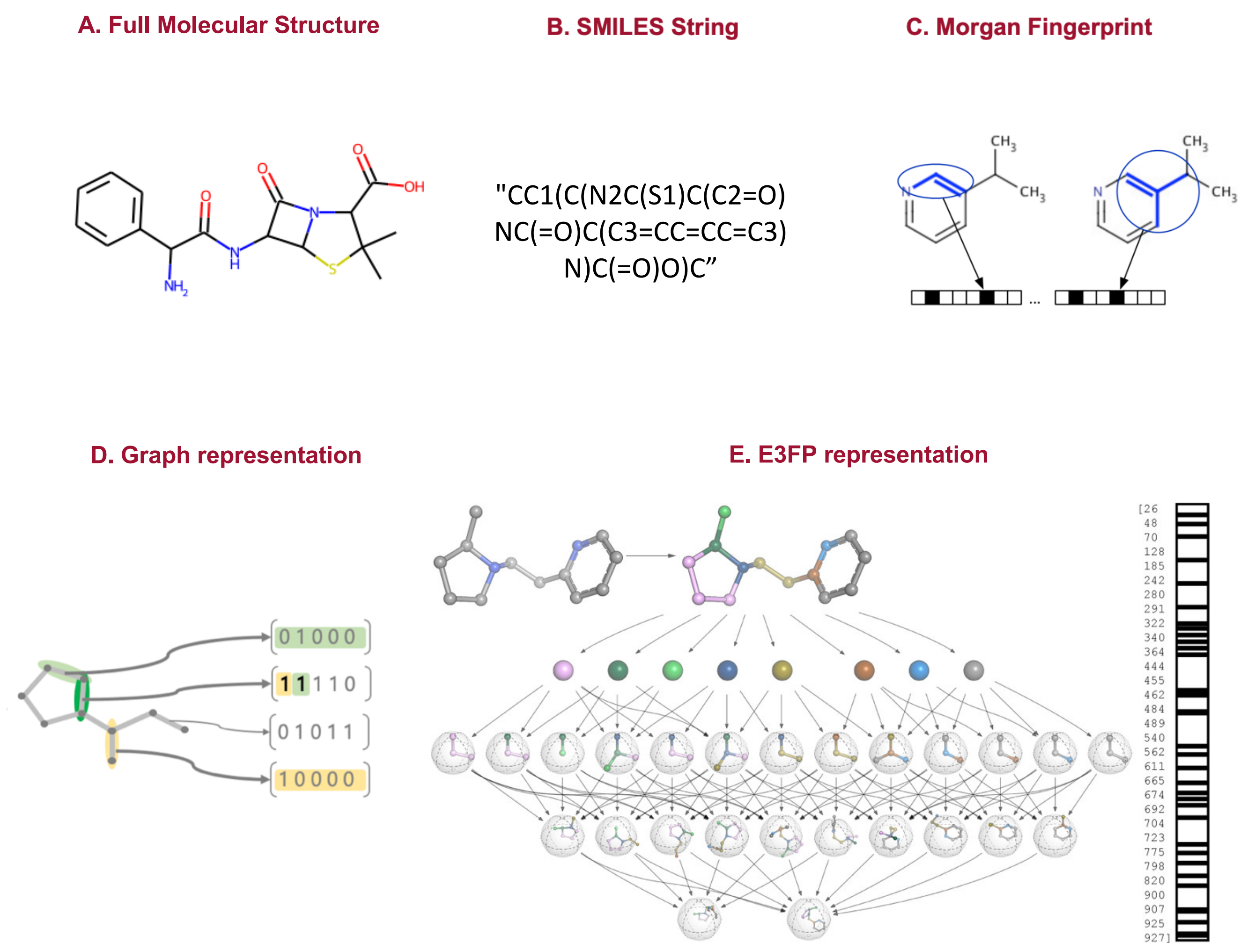


Figure 3 - Side-by-Side Comparison of Various Molecular Representation Methods. **A)** An example of a full molecular structure shows the structure of the antibiotic ampicillin (generated using the Python RDKit package's molecular visualization functions). **B**) An example of a SMILES string representation shows the SMILES string of the antibiotic ampicillin [42]. **C)** A Morgan Fingerprint is created by hashing substructures to bits (reproduced from [18]). **D)** Molecular graphs represent atoms and bonds as nodes and edges (graphic created by Jacqueline Valeri). **E)** E3FP fingerprints represent substructures in 3D as bits (reproduced from [41]). While SMILES strings are more directly interpretable by a user, Morgan Fingerprints, graph representations, and E3FP representations are able to vectorize molecular substructures such that these vector representations can be used by a variety of neural network architectures.

In addition to these molecular representation inputs, another component of the inputs to all three models is a set of 200 RDKit global molecular features. This is a set of 200 additional features (such as molecular weight, the number of partial charges, the number of aromatic rings, etc.) that the RDKit Python package calculates for each given molecular structure. The decision was made to include these 200 RDKit features in addition to the molecular representations, as these features provide additional information to the models regarding molecular features that may potentially influence synergistic interactions between the two halves of the combination.

The ground truth outputs are a set of binary synergy scores [see Section 2.3]. The decision was made to use binary synergy scores instead of continuous scores (such as Bliss synergy scores), as the goal was to create a classification machine learning model rather than a regression model. The reason for this is that, as previous work in the Collins Lab has demonstrated, regression models perform with extremely poor accuracy when making predictions regarding antibiotics, especially combination antibiotic therapies. For this reason, the decision was made to create classification models since they are of greater utility to experimentalists, and by extension, the decision was made to use binary ground truth outputs.

4.3 Machine Learning Model Architecture

There are three main machine learning model architecture designs that were tested in this project, each with their own unique set of advantages and disadvantages (Table 2). Specifically, the three model types tested were a feed forward neural network with 2-dimensional inputs (2D FFN), a graph neural network (GNN), and a feed forward neural network with 3-dimensional inputs (3D FFN).

First, the three model architectures differ with regards to the type of input molecular representations required. That is, 2D FFNs require Morgan Fingerprints, GNNs require molecular graphs, and 3D FFNs require E3FP representations. Consequently, due to differences in the degree of structural information captured by these molecular representation methods, the models themselves by extension are also able to consider differing degrees of structural information while making predictions (Table 2).

This in turn has downstream impacts on model accuracies and computational efficiencies. Given that the 3D FFN accounts for the largest amount of structural information out of the three model architectures, it also has the potential for the greatest accuracy (Table 2). In contrast, the 2D FFN accounts for the least structural information out of the three model architectures. While this lowers its potential for achieving equivalent accuracy to the other two model architectures, it also grants the 2D FFN the advantage of having the greatest potential for computational efficiency (Table 2).

Table 2. Comparison of the advantages and disadvantages of the 2D FFN, GNN, and 3D FFN model architectures.

	2D FFN	GNN	3D FFN
Capturing structural features	✓	✓✓	✓✓✓
High Accuracy	✓	✓✓	✓✓✓
Computational Efficiency	✓✓✓	✓✓	✓

Due to these differing advantages and disadvantage of the three model architectures, it was crucial to identify a model design that best balances these factors in order to meet the technical specifications as closely as possible. For this reason, designs for all three model types were created and tested.

4.3.1 2D Feed Forward Neural Network (2D FFN)

A Feed Forward Neural Network (FFN) is one of the simplest types of neural networks, and in an FFN, information propagates in one direction [43]. That is, the input information is passed into the input layer of an FFN, from which it passes in the same direction to the hidden layers. The hidden layers are the layers in which the model uses an algorithm to learn to convert the inputs to the outputs. For example, if a hidden layer is using a linear activation function, every node in the FFN hidden layer performs the $y = mx + b$ operation, where x is the input into that node and y is the output. Over the course of model training, the optimal values of m and b for each node are learned by the hidden layer to generate outputs that best match the ground-truth. Finally, from the hidden layers, information is passed to the output layers.

As discussed above (Table 2), an advantage of FFNs is that the simplicity of their architecture usually ensures that they are not as computationally intensive as the other types of models. Thus, an FFN model architecture is advantageous for meeting the computational intensiveness technical specifications (Table 1). FFNs are the simplest type of neural network, which offers several advantages but can also sometimes limit the model accuracy. Thus, a potential drawback of using an FFN is that it may be too simple to fully capture the complexity of the combination molecular structure and the complexity in predictions using these structures. However, based on existing work by Yang et al. [13], an auROC of ~0.7-0.8 is expected for the FFNs. auPR values seen by Yang et al. [13] were more variable, but an average auPR of ~0.1-0.2 is expected for FFNs. Thus, an FFN design has the potential to meet the accuracy-related technical specifications (Table 1).

There are a number of important parameters in the 2D FFN architecture design, for which design choices were made. For many parameters in machine learning models, the optimal choice for the parameter is difficult to obtain through intuition or calculations. Rather, model parameters are often chosen by hyperparameter optimization algorithms.

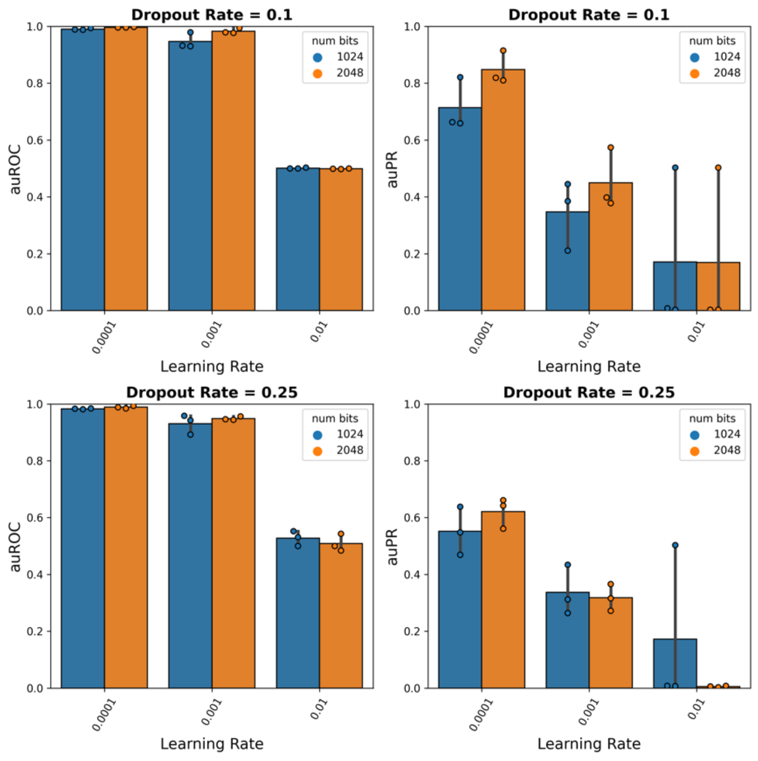
Thus, a grid search hyperparameter optimization was performed to identify the optimal choices for these parameters:

1. Dropout rate, the rate at which randomly selected neurons in the neural network are ignored while training. Dropout is used to prevent overfitting of models.
2. Learning rate, the rate of adaptation of the model to the error in each iteration. It is important to achieve a balance between a learning rate that is too low (which causes the model to train very slowly) and a learning rate that is too high (which may cause the model to overshoot an optimal configuration).
3. Number of bits in the Morgan Fingerprint inputs. As previously discussed, Morgan Fingerprints are generated by hashing substructures to bits. Increasing the number of bits in the Morgan Fingerprint (up to a certain point), may increase the information available to the model, thereby potentially improving accuracy.

The hyperparameter optimization was performed by first identifying the values to be tested for each of these three parameters. In order to capture the range of reasonable values, dropout rates of 0.1 and 0.25; learning rates of 0.01, 0.001, and 0.0001; and 1024 and 2048 Morgan Fingerprint bits were tested in this hyperparameter optimization. The optimization itself was performed by iterating through all possible combinations of these choices for each parameter, i.e. by training and testing multiple 2D FFN models, each of which used a different combination of these parameter choices. For each combination of parameter choices, three optimization runs were performed, meaning that three individual models making up an ensemble were trained using that same combination of parameter choices, and each model was trained and tested for 30 epochs.

After performing the hyperparameter optimization, the two most important accuracy metrics, the auROC and auPR, were used to compare the performances of models using the different parameter choices (Figure 4). Slight differences in model performances can be seen between the three learning rate options, the two dropout rate options, and the two Morgan Fingerprint bit number options. In both the training set and the testing set, models with learning rates of 0.0001 generally demonstrate the highest auROC and auPR metrics. With regards to dropout rate, although differences between the two dropout rates are not as clear, models with a dropout rate of 0.1 seems to demonstrate improved performance in many cases. Likewise, with regards to the number of Morgan Fingerprint bits, there are not large differences in model performances, but in many cases, models run with 2048-bit Morgan Fingerprints seem to demonstrate slightly improved performance.

A) 2D FFN Hyperparameter Optimization Results for Training Set



B) 2D FFN Hyperparameter Optimization Results for Testing Set

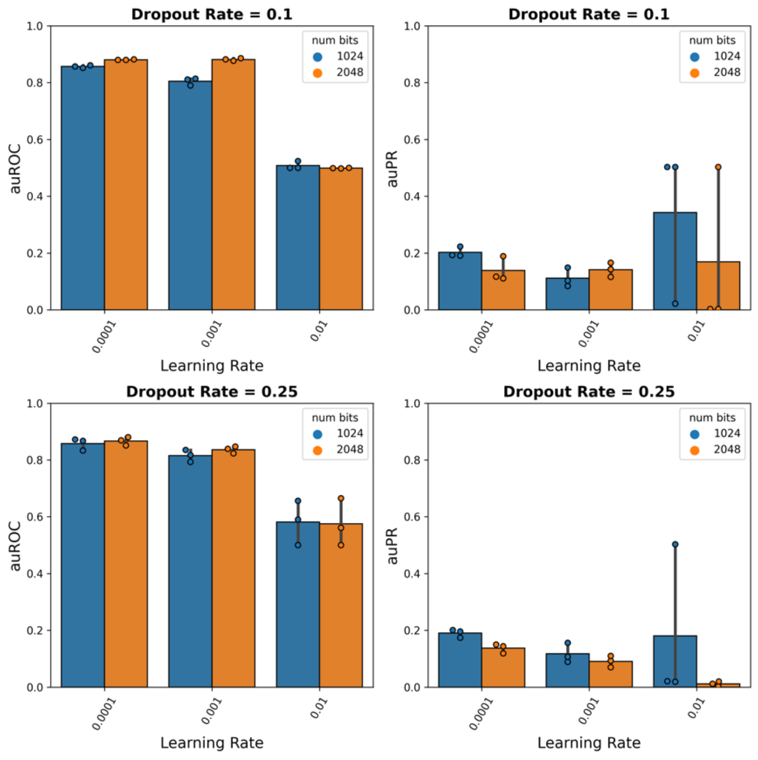


Figure 4 - Results of Hyperparameter Optimization of the 2D FFN. The hyperparameter optimization results for the **A)** training set (n=3) and **B)** testing set (n=3) demonstrate differences in model auROC and auPR

performance metrics as learning rate, dropout rate, and the number of bits in the input Morgan Fingerprints are varied. Significant differences in performance metrics were observed (using one-tailed Mann-Whitney U test) between the three learning rates in both the training and testing sets as well as between the two bit numbers in the testing set (see Table 3). The results support the design choices to use a learning rate of 0.0001, dropout rate of 0.1, and 2048 bits in the input Morgan Fingerprints. *(*Note: a learning rate of 0.01 results in an auPR of 0.5 for a few of models due to artifacts in Scikit-learn's method of generating PR curves when the model predicts all negatives)*

To further elucidate the best choice for each of these three parameters, one-tailed Mann-Whitney U significance tests were performed across all model runs for each individual parameter choice (Table 3).

Table 3. Significance tests for 2D FFN hyperparameter optimization

	Training Set (p values)		**Testing Set (p values)**	
	auROC	**auPR**	**auROC**	**auPR**
dr = 0.1 vs dr = 0.25	0.24196	0.12714	0.3707	0.10935
lr = 0.01 vs lr = 0.001	< 0.00001*	0.01321*	< 0.00001*	0.08692
lr = 0.01 vs lr = 0.0001	< 0.00001*	0.00004*	< 0.00001*	0.08692
lr = 0.001 vs lr = 0.0001	0.00014*	0.00004*	0.01618*	0.001*
1024 bits vs 2048 bits	0.06426	0.48803	0.08379	0.02872*

dr = dropout
lr = learning rate
** = statistically significant at the p < 0.05 threshold using one-tailed Mann-Whitney U test*

As can be seen, there are generally statistically significant differences in the three learning rate choices, which reinforce the observation from Figure 4 that 0.0001 is the most effective learning rate out of the three. It is important to note an anomaly when using a learning rate of 0.01. As shown in Figure 4, when using this learning rate, auPR values for a few models are occasionally 0.5. Generally, high auPR values indicate stronger performance, but in this case, an auPR of 0.5 is indicative of extremely poor performance. A model that has very poor or no predictive capabilities sometimes predicts only one class, and the case of this model, it predicts all negatives. PR curves in this project were plotted using the precision_recall_curve function in Python's Scikit-learn package [36]. Normally, this function generates a PR curve by creating a range of thresholds and plotting the precision and recall for predicted scores above each threshold [36]. However, if the model predicts all negatives (all 0's), the only threshold that can be created is 0. Given that this is an impossible scenario to plot, the function defaults to only plotting the data points [1,0] and [0,1]. This artificially plots a diagonal line with an area under the curve of 0.5. Thus, these few cases in which a learning rate of 0.01 results in an auPR of 0.5 are not indicative of strong performance but rather signify extremely poor performance.

As observed in Figure 4, the differences between dropout rates of 0.1 and 0.25 and between bit numbers of 1024 and 2048 generally fall short of the threshold for statistical significance, indicating that the exact choices of dropout rate and bit number do not significantly impact model performance. However, given that a dropout rate of 0.1 and bit number of 2048 sometimes result in improved performance (Figure 4), these are the better choices for these parameters.

In addition to these three parameters, another design choice involved selecting an activation function for the final output layer of the 2D FFN. While the "ReLu" activation functions of the hidden layers are fairly standard, the activation function applied to the output layer can drastically change model outputs, and a variety of activation functions are often used for output layers in the field. In this case, the decision was made to use the sigmoid activation function, as this function returns values between 0 and 1, where values closer to 1 indicate a greater probability of being a synergistic antibiotic combination. Such output values are easily interpretable for users as probability of synergy, which improves the utility of the predictions for the user. For this reason, the decision was made to use this activation function.

The final design choice was the threshold for considering output values as positives, i.e. as combinations that demonstrate antibacterial synergy. While classification models typically select 0.5 as the logical threshold, past results demonstrate that this is not actually the optimal inflection point separating positive results from negative results. Specifically, past results from molecular models created by Collins Lab scientists have demonstrated that when output scores are plotted against the ranks or frequencies of these scores, a natural inflection point consistently occurs around 0.1 (Figure 5). Generally, antibiotic data sets often demonstrate class imbalance, with the majority of the molecules in antibiotic data sets being negatives. Likewise, in these plots, the scores with the greatest frequencies occur below ~0.1, while the less frequent scores occur above 0.1 (Figure 5). This indicates that the majority of the molecules in the dataset, i.e. most of the negatives, score below 0.1, while the sparser set of positives score above 0.1.

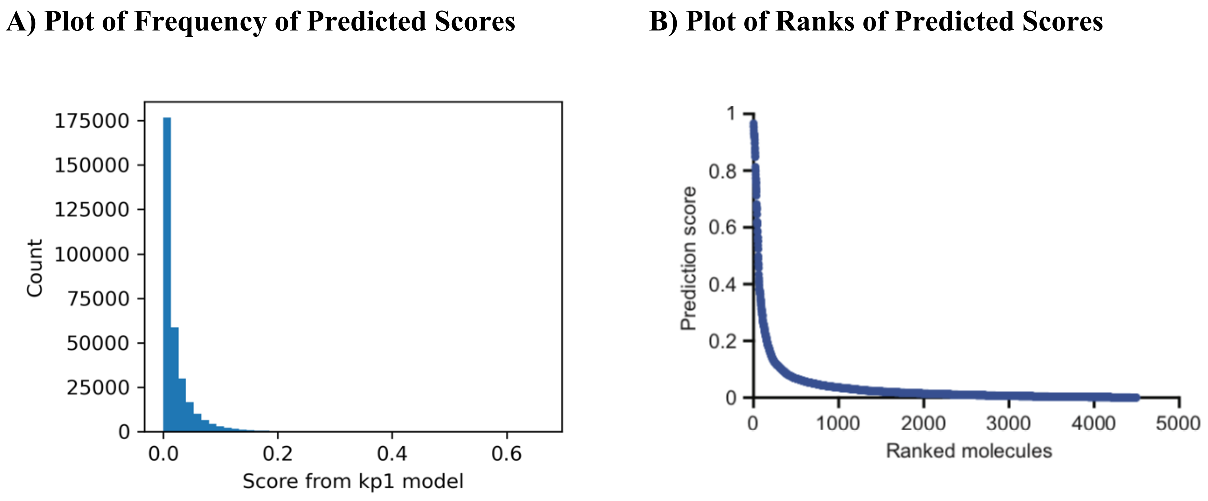


Figure 5 – Frequency of Predicted Scores by Other Molecular Models. A) The plot of the frequency of scores predicted by a model previously created by Collins Lab scientists that was run on combinations from the Charles River molecular library shows an inflection point around a score of 0.1. **B)** The plot of the ranks of scores predicted by the Chemprop model created by Stokes et al. when run on a dataset of single-agent molecules [10] also shows an inflection point near a predicted score of 0.1. Both plots support the choice of defining positives as combinations receiving a predicted score of ≥ 0.1 and negatives as combinations receiving a predicted score of < 0.1.

For this reason, 0.1 was selected as the optimal threshold, supported by prior precedence, to classify molecules as positives vs. negatives. Using this threshold, combinations receiving a predicted synergy score of ≥ 0.1 are classified as positives while combinations receiving a predicted score of < 0.1 are classified as negatives.

In summary, the final set of design choices for the 2D FFN are as follow:
- Learning rate of 0.0001
- Dropout rate of 0.1
- 2048 bits in the Morgan Fingerprints that serve as inputs to the model
- Sigmoid function as the activation function of the final output layer of the model
- 0.1 as the threshold for classifying outputs as positives (synergistic combinations) vs. negatives (non-synergistic combinations)

4.3.2 Graph Neural Network (GNN)

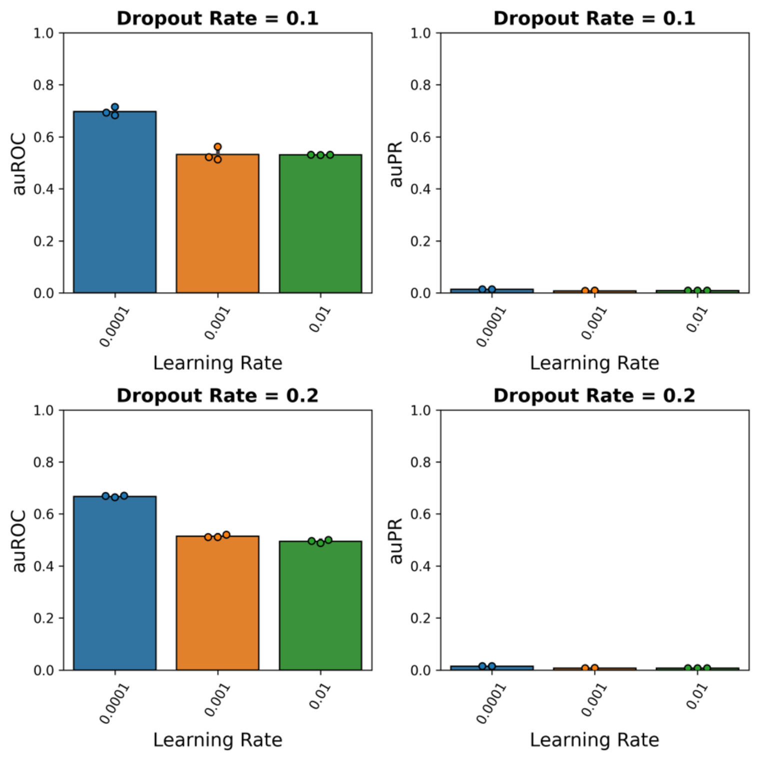
A Graph Neural Networks (GNN) is a more complex type of neural network than the FFN. As previously described, GNNs are especially useful for molecular machine learning because they have a message-passing structure that allows for the propagation of information within each local neighborhood of atoms. GNNs represent the atoms and bonds in compounds as networks of nodes and edges.

As discussed above (Table 2), the biggest advantage of GNNs is that the molecular graph representations capture more structural information than Morgan Fingerprint representations. Thus, a GNN model architecture can be advantageous for meeting the accuracy technical specifications (Table 1). However, due to the complexity of GNNs, a potential drawback is that the GNN may be more computationally intensive than the 2D FFN. Existing work by Stokes et al. [10], has demonstrated an auROC of ~0.896 when using the Chemprop GNN. Thus, a GNN design could have the potential to provide an improvement over the 2D FFN in terms of meeting the accuracy-related technical specifications (Table 1).

There are once again a number of important parameters for the GNN architecture design, for which optimal values were identified through hyperparameter optimization. Two choices for the dropout rate (0.1 and 0.2) and three choices for the learning rate (0.01, 0.001, 0.0001) were tested through the hyperparameter optimization.

After performing the hyperparameter optimization, the auROC and auPR were again used to compare the performances of models run using the different parameter choices (Figure 6). In both the training set and the testing set, models with learning rates of 0.0001 seem to generally demonstrate the highest auROC and auPR metrics. With regards to dropout rate, although the differences between the two dropout rates are again not as clear, models run with a dropout rate of 0.1 seems to demonstrate slightly improved performance in a few cases. With all combinations of parameter choices, model performance in the testing set is especially poor, as the auROC values never exceed approximately 0.6 and auPR values also remain extremely low.

A) GNN Hyperparameter Optimization Results for Training Set



B) GNN Hyperparameter Optimization Results for Testing Set

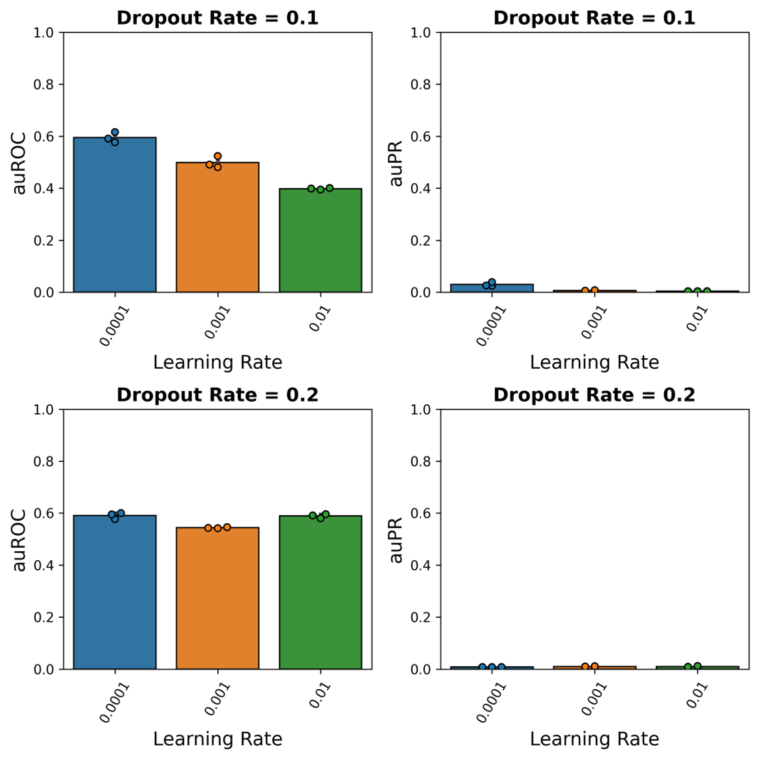


Figure 6 - Results of Hyperparameter Optimization of the GNN. The hyperparameter optimization results for the **A)** training set (n=3) and **B)** testing set (n=3) demonstrate differences in model auROC and auPR metrics

as learning rate and dropout rate are varied. Significant differences in performance metrics were observed (using one-tailed Mann-Whitney U test) between two of the three learning rates as well as between the dropout rates (see Table 4). The results support the design choices to use a learning rate of 0.0001 and dropout rate of 0.1.

One-tailed Mann-Whitney U significance tests were once again performed across all model runs for each individual parameter choice (Table 4).

Table 4. Significance tests for GNN hyperparameter optimization

	Train		**Test**	
	auROC	**auPR**	**auROC**	**auPR**
dr = 0.1 vs dr = 0.2	0.03216*	0.12507	0.02619*	0.15386
lr = 0.01 vs lr = 0.001	0.34458	0.43644	0.46812	0.18943
lr = 0.01 vs lr = 0.0001	0.00256*	0.00256*	0.07493	0.08692
lr = 0.001 vs lr = 0.0001	0.00256*	0.00256*	0.00256*	0.13136

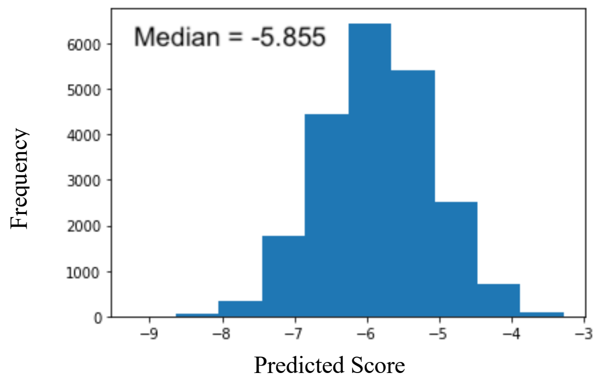
dr = dropout
lr = learning rate
** = statistically significant at the $p < 0.05$ threshold using one-tailed Mann-Whitney U test*

In Table 4, these parameters have fewer statistically significant differences, indicating that the exact choice of these parameters has less of an impact on the GNN performance. However, there are often significant differences in the three learning rate choices, which reinforce the observation from Figure 6 that 0.0001 is the most effective learning rate out of the three. As observed in Figure 6, the differences between dropout rates of 0.1 and 0.2 are minor and not always consistent in direction, nor are the differences always significant. However, given that a dropout rate of 0.1 does sometimes result in improved performance (Figure 6), the choice can be made to use a dropout rate of 0.1. Furthermore, as discussed previously, testing set performance is especially poor for this model, suggesting that the model may have difficulties with generalizing. However, the dropout rate of 0.1 and learning rate of 0.0001 aid the model in achieving the best testing set performance out of all other parameter choice combinations, once again suggesting that they are the optimal parameters for achieving the best possible performance in both the training and testing sets.

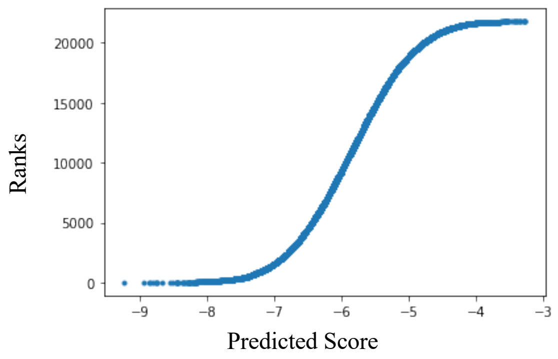
In addition to these parameters, another design choice was activation function selection for the final output layer of the GNN. While the 2D FFN used the sigmoid activation function, this resulted in poorer performance with the GNN. In contrast, the log_softmax activation function is better suited to the GNN. This log_softmax function returns continuous values between negative infinity and 0, where values closer to 0 indicate a greater probability of being a synergistic antibiotic combination. Thus, the outputs from this function can once again be interpreted by a user as probability of success.

Finally, the last design choice was regarding the threshold for considering output values from this log_softmax function as positives. While past results from molecular models created by Collins Lab scientists have demonstrated reasonable thresholds for outputs from the sigmoid function (Figure 5), it was necessary to establish a similar threshold for the outputs from this log_softmax function. To do so, histograms of the scores predicted by the GNN (with dropout rate of 0.1 and learning rate of 0.0001 as identified through the hyperparameter optimization) were created. These histograms revealed that the distribution of predictions in both training and testing sets are relatively normal (Figure 7). Then, the predicted scores were plotted against their ranks to identify inflection points separating positive outputs from negative outputs, as in Figure 5. Consistent inflection points were observed in these plots in both the training set and the testing set, generally in the region of the median of the scores (Figure 7). Since the histograms revealed that the distribution of predictions was relatively normal, the medians of the training and testing sets are reasonable thresholds for distinguishing positives from negatives.

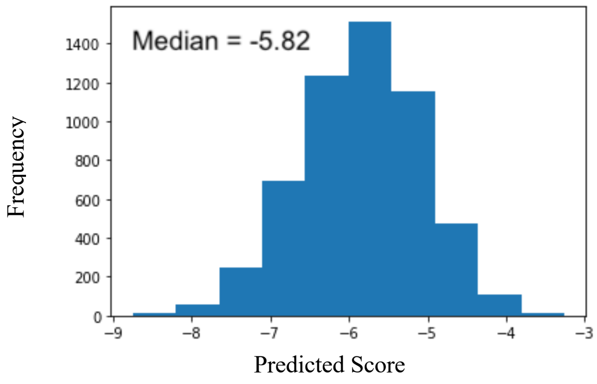
A)



B)



C)



D)

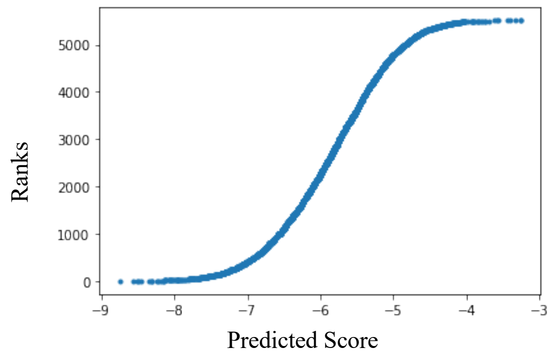


Figure 7 – Distribution and Ranks of Scores Predicted by the GNN. A) The plot of a histogram of the scores predicted by the best-performing GNN design for the training set indicates that the predictions are relatively normally distributed, with a median of -5.855. **B)** The plot of the ranks of scores predicted by the best-performing GNN design for the training set supports the choice of defining positives as molecules receiving a predicted score of ≥ (median predicted score) and negatives as molecules receiving a predicted score of < (median predicted score). **C)** The plot of a histogram of the scores predicted by the best-performing GNN design for the testing set indicates that the predictions are relatively normally distributed, with a median of -5.82. **D)** The plot of the ranks of scores predicted by the best-performing GNN design for the testing set supports the choice of defining positives as molecules receiving a predicted score of ≥ (median predicted score) and negatives as molecules receiving a predicted score of < (median predicted score).

For this reason, the median of the predicted scores output by the GNN is the optimal threshold to classify combinations as positives vs. negatives. Using this threshold, combinations receiving a predicted synergy score of ≥ (median predicted score) are classified as positives while combinations receiving a predicted score of < (median predicted score) are classified as negatives.

In summary, the final set of design choices for the GNN are as follow:

- Learning rate of 0.0001
- Dropout rate of 0.1
- Log_softmax function as the activation function of the final output layer of the model
- The median of the predicted scores as the threshold for classifying outputs as positives (synergistic combinations) vs. negatives (non-synergistic combinations)

4.3.3 3D Feed Forward Neural Network (3D FFN)

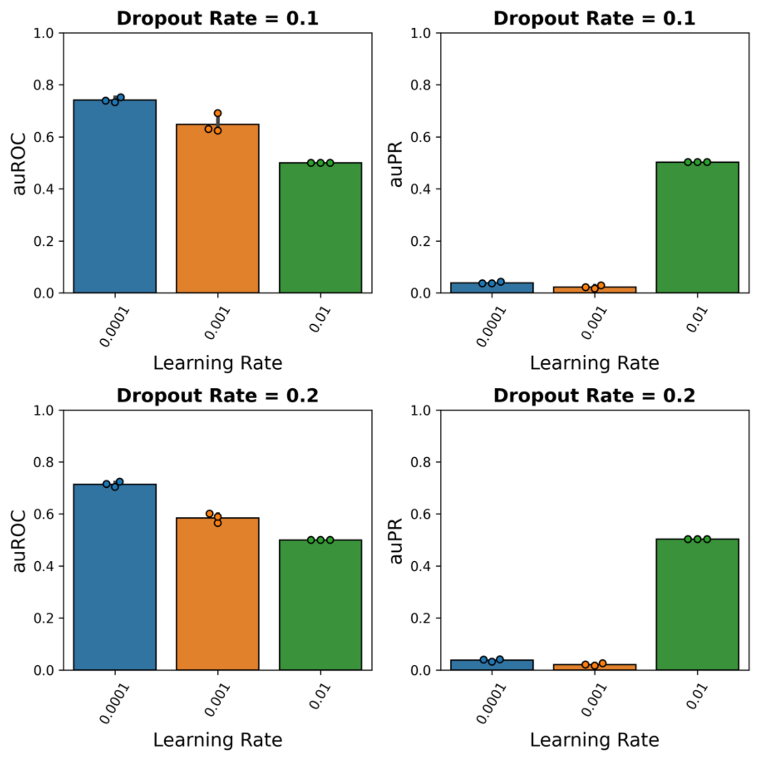
A 3D Feed Forward Neural Network (3D FFN) has the same overall architecture as the 2D FFN, as both are fundamentally FFNs. The difference in this case is that the inputs, rather than being 2D Morgan Fingerprints, are 3D E3FP fingerprints. Thus, the different advantages and disadvantages of an FFN architecture (described in Section 4.3.1) also apply to the 3D FFN. However, the 3D FFN has an additional advantage in that its inputs provide additional structure information in three dimensions. Thus, the 3D inputs in the 3D FFN may aid the model in better meeting the accuracy technical specifications.

The first design decision to be made was the number of bits in the E3FP fingerprints that serve as inputs for the 3D FFN. Given that the process of generating E3FPs is extremely computationally- and time-intensive, testing different bit numbers as was done for the Morgan Fingerprints in Section 4.3.1 was not feasible. Hence, since 2048 bits proved to be the optimal choice for the Morgan Fingerprints, the decision was made to use 2048 bits when generating each individual molecule's E3FP as well.

Like with the 2D FFN, a hyperparameter optimization was performed to identify the optimal design choices for the dropout rate and the learning rate for this 3D FFN. Once again, two choices for the dropout rate (0.1 and 0.2) and three choices for the learning rate (0.01, 0.001, 0.0001) were tested through the hyperparameter optimization.

The auROC and auPR were again used to compare the performances of models run using the different parameter choices (Figure 8). In both the training set and the testing set, models run with learning rates of 0.0001 seem to demonstrate the highest auROC and auPR metrics. The impact of dropout rate is again not as significant, but models with a dropout rate of 0.1 seems to demonstrate slightly improved performance in a few cases.

A) 3D FFN Hyperparameter Optimization Results for Training Set



B) 3D FFN Hyperparameter Optimization Results for Testing Set

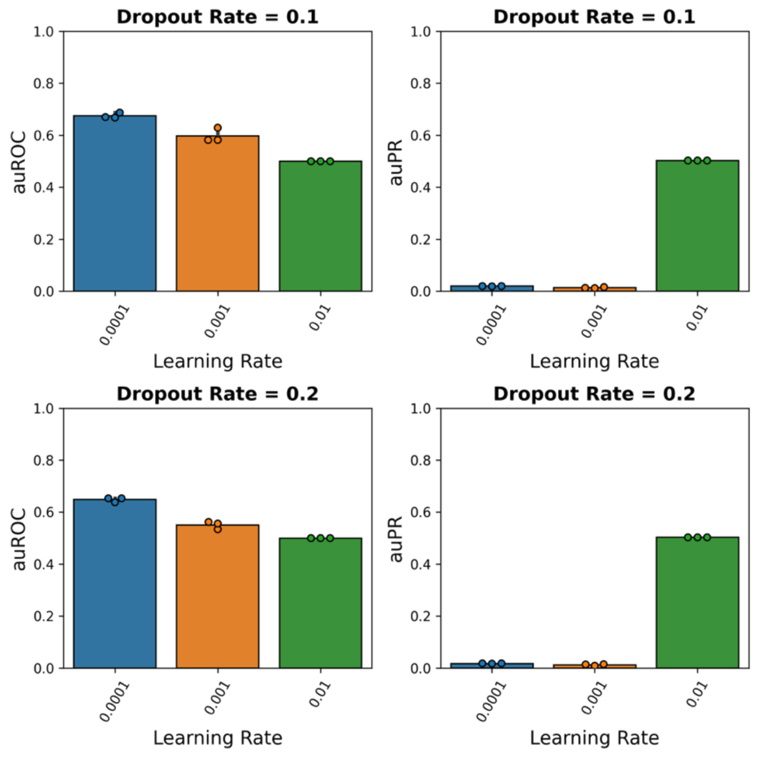


Figure 8 - Results of Hyperparameter Optimization of the 3D FFN. A The hyperparameter optimization results for the **A)** training set (n=3) and **B)** testing set (n=3) demonstrate differences in model auROC and auPR

metrics as learning rate and dropout rate are varied. Significant differences in performance metrics were observed (using one-tailed Mann-Whitney U test) between the three learning rates (see Table 5). The results support the design choices to use a learning rate of 0.0001 and dropout rate of 0.1. *(*Note: a learning rate of 0.01 results in an average auPR of 0.5 due to artifacts in Scikit-learn's method of generating PR curves when the model predicts all negatives)*

Like with the 2D FFN and GNN, one-tailed Mann-Whitney U significance tests were once again performed across all model runs for each individual parameter choice (Table 5).

Table 5. Significance tests for GNN hyperparameter optimization

	Train		Test	
	auROC	**auPR**	**auROC**	**auPR**
dr = 0.1 vs dr = 0.2	0.22663	0.46414	0.22663	0.34458
lr = 0.01 vs lr = 0.001	0.00256*	0.00256*	0.00256*	0.00256*
lr = 0.01 vs lr = 0.0001	0.00256*	0.00256*	0.00256*	0.00256*
lr = 0.001 vs lr = 0.0001	0.00256*	0.00256*	0.00256*	0.00256*

dr = dropout
lr = learning rate
** = statistically significant at the p < 0.05 threshold using one-tailed Mann-Whitney U test*

As can be seen, there are statistically significant differences in the three learning rate choices, confirming that 0.0001 is the most effective learning rate by a significant margin. Like with the 2D FFN, an anomaly occurs when using a learning rate of 0.01. As shown in Figure 8, when using this learning rate, auPR values are on average close to 0.5 but in reality, indicate extremely poor performance. Like with the 2D FFN, this anomaly occurred due to artifacts created by Scikit-learn's precision_recall_curve function when a model outputs all negatives (see Section 4.3.1 for more explanation). Thus, the learning rate of 0.01 and 0.001 both demonstrate poorer performance than the learning rate of 0.0001, suggesting that 0.0001 is the optimal choice.

The differences between dropout rates of 0.1 and 0.2 miss the threshold for statistical significance (Figure 8), indicating that the exact choice of dropout rate does not significantly impact model performance. However, given that a dropout rate of 0.1 sometimes results in improved performance (Figure 8), it is likely the better out of the two choices for this parameter. Testing set performance is again generally poor for this model, suggesting that the model may, like the GNN, have difficulties with generalizing. However, the dropout rate of 0.1 and learning rate of 0.0001 aid the model in achieving the best testing set performance out of all other parameter combinations, reinforcing their selection as the optimal parameters for achieving the best possible performance in both the training and testing sets.

In addition to these parameters, the same sigmoid activation function used with the 2D FFN is used with the 3D FFN since the two models have the same fundamental architecture. Since the same sigmoid activation function is used, the same threshold for considering output values as positives (0.1) is used as well.

In summary, the final set of design choices for the 3D FFN are as follow:

- Learning rate of 0.0001
- Dropout rate of 0.1
- 2048 bits in the E3FP fingerprint of each individual molecule
- Sigmoid function as the activation function of the final output layer of the model
- 0.1 as the threshold for classifying outputs as positives (synergistic combinations) vs. negatives (non-synergistic combinations)

4.4 User Interface Design

The user-friendliness of the user interface used to deliver the model to the clients is critical to this design. Usability and accessibility are crucial parts of best practices in the field of machine learning, as described by the Fairness, Accountability, and Transparency in Machine Learning (FAT ML) guidelines [44] (Appendix 1). Thus, it is important that the user interface created in this project best suits the needs and expertise of Collins Lab experimentalists, the intended clients for this project.

The final user interface in this project is a Jupyter Notebook that allows experimentalists to use this final model to make predictions regarding the synergistic antibacterial activities of combination therapies in their own datasets (Figure 9). A graphical user interface could theoretically be created but given the relative computational confidence of experimentalists in the home departments of the Collins and Blainey labs, such an interface is not necessary. Experimentalists in the Collins Lab, for example, have had specific training in using Jupyter Notebooks through the MIT Biological Engineering department's core classes 20.420 and 20.440. Thus, the target population can feasibly use this model through a Jupyter Notebook.

The notebook must offer the user the opportunity to perform 3 key tasks that are most useful to experimentalists (Figure 9):

1) Assess the synergy of specified molecular combinations:

   The user can upload a list of SMILES string representations of either the separate agents in combinations or the full combinations they wish to assess or validate. Given that the dataset is formatted in a manner compatible with the notebook, the Jupyter Notebook processes these SMILES strings and uses the machine learning model to make predictions regarding the synergy of each combination. The output to the user is a list of predictions for each combination indicating the probability of the combination demonstrating synergistic activity.

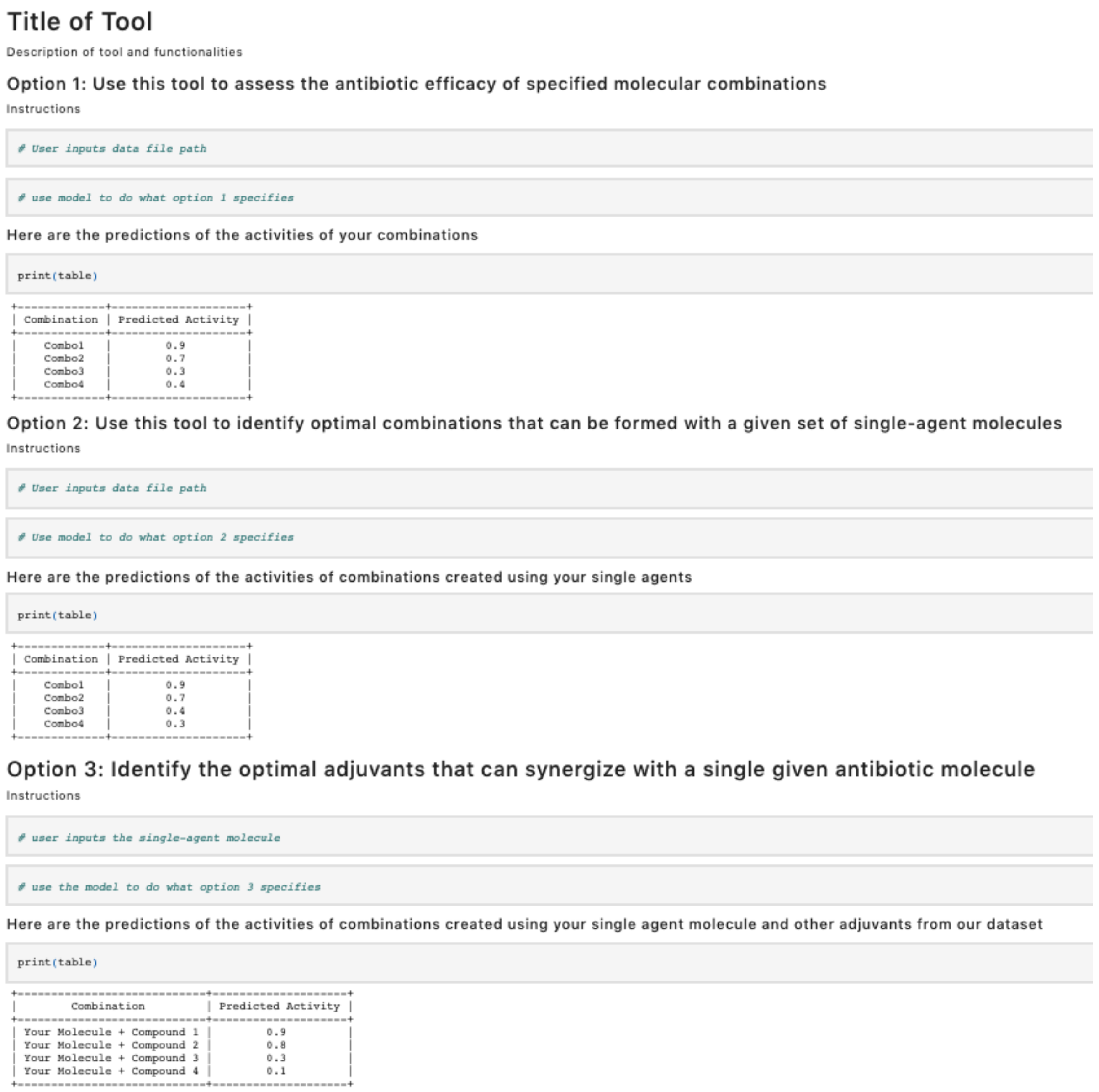
2) Identify optimal combinations that can be formed with a given set of single-agent molecules

   The user can upload a list of SMILES string representations of the single-agent molecules that they wish to use to form pairwise combination therapies. The Jupyter Notebook first processes these SMILES strings and creates pairwise combinations from all possible pairings of single-agent molecules in the input dataset, excluding pairings of molecules with themselves. Then, the Jupyter Notebook uses the machine learning model to make predictions regarding the synergy of each pairwise combination of molecules from the dataset. The output to the user is a list of pairwise combinations composed of the single-agent molecules that were input by the user along with predictions regarding their synergies. Since the model outputs continuous values indicative of probability, the list of pairwise combinations output to the user is sorted by the synergy scores, such that the combinations in which the model has the most confidence appear first.

3) Identify the optimal adjuvants that can synergize with a single given antibiotic molecule

   The user can enter the SMILES string representation of a single antibiotic molecule. On the backend, combinations of this antibiotic with all possible adjuvants are created, with the available adjuvants being those with which the machine learning model was originally trained. These combinations then serve as inputs into the model, which predicts the synergy of the combinations. Then, the combinations will be output to the user, along with the predicted synergy scores.

Figure 9 below shows a high-level schematic of the design of this Jupyter Notebook user interface.



Title of Tool

Description of tool and functionalities

Option 1: Use this tool to assess the antibiotic efficacy of specified molecular combinations

Instructions

```
# User inputs data file path
```

```
# use model to do what option 1 specifies
```

Here are the predictions of the activities of your combinations

```
print(table)
```

```
+-------------+--------------------+
| Combination | Predicted Activity |
+-------------+--------------------+
|    Combo1   |        0.9         |
|    Combo2   |        0.7         |
|    Combo3   |        0.3         |
|    Combo4   |        0.4         |
+-------------+--------------------+
```

Option 2: Use this tool to identify optimal combinations that can be formed with a given set of single-agent molecules

Instructions

```
# User inputs data file path
```

```
# Use model to do what option 2 specifies
```

Here are the predictions of the activities of combinations created using your single agents

```
print(table)
```

```
+-------------+--------------------+
| Combination | Predicted Activity |
+-------------+--------------------+
|    Combo1   |        0.9         |
|    Combo2   |        0.7         |
|    Combo3   |        0.4         |
|    Combo4   |        0.3         |
+-------------+--------------------+
```

Option 3: Identify the optimal adjuvants that can synergize with a single given antibiotic molecule

Instructions

```
# user inputs the single-agent molecule
```

```
# use the model to do what option 3 specifies
```

Here are the predictions of the activities of combinations created using your single agent molecule and other adjuvants from our dataset

```
print(table)
```

```
+---------------------------+--------------------+
|        Combination        | Predicted Activity |
+---------------------------+--------------------+
| Your Molecule + Compound 1 |        0.9         |
| Your Molecule + Compound 2 |        0.8         |
| Your Molecule + Compound 3 |        0.3         |
| Your Molecule + Compound 4 |        0.1         |
+---------------------------+--------------------+
```

Figure 9 - High level schematic of the Jupyter Notebook Interface. This interface is aimed at allowing experimentalists to use the machine learning model to aid their efforts. The interface provides the user the option to perform three key tasks, while inputting no more information than the file path to the dataset they wish to analyze or the single molecule itself in the third task.

5 Building/Prototyping

5.1 Key Tools and Materials

Three main groups of tools were used for the building and prototyping in this project: one set of tools for generating molecular representations, a second set of tools for model creation, and a third set of tools for running and testing the models.

To generate molecular representations to serve as inputs for the models, packages used included the following:

1. RDKit, an open-source package used for cheminformatics analysis [29], was used for generating Morgan Fingerprints and the 200 global molecular features for each combination.
2. Mol2vec, an un-supervised machine learning approach for converting molecular structures to vectors [15], was used for generating molecular graphs for each combination.
3. E3FP, a package that integrates with RDKit for generating three-dimensional molecular fingerprints [41], was used for generating E3FP representations of each combination.

The Python PyTorch package [45] was used to create the models. To train and test the models, a personal computer CPU was used for less intensive computations and models (2D FFN), and Google Colab GPU was used for more intensive computations and models (GNN and 3D GNN).

5.2 Input Dataset Setup

Separate input datasets were assembled for each of the three model types being tested. For the 2D FFN, as mentioned previously in Section 4.2, Morgan Fingerprints are the appropriate input molecular representation type. As described in Section 4.3.1, the hyperparameter optimization results suggest the use of 2048-bit Morgan Fingerprints. Thus, the SMILES string representations of the combinations were converted to 2048-bit Morgan Fingerprints using built-in functions in the RDKit Python library [29]. Additionally, 200 RDKit molecular features were also generated for the combinations. When appended together to create a final single input vector, these 2048-bit Morgan Fingerprints and 200 global molecular features together result in a 2248-length input vector. The 2248-length input vectors corresponding to each combination, along with their respective binary synergy score outputs, make up the complete 2D FFN ground truth dataset. In order to train, test, and validate the model, this dataset was split into three separate datasets. The training set is the portion of the ground truth data from which the models learn to make predictions, while the testing set is a separate portion of the ground truth data that is used for testing and evaluating the models. Following usual practices, the ground truth dataset was split 80% for training, 10% for testing, and 10% for validation.

As mentioned in Section 4.2, the input molecular representations for the GNN are molecular graphs. Thus, the SMILES string representations of the combinations were converted to molecular graphs that were 75 elements long for each atom using Mol2vec, an unsupervised machine learning approach for converting molecular structures to vectors [15]. These molecular graphs, along with the 200 RDKit global molecular features, comprise the inputs to the GNN. These inputs along with the binary synergy scores make up the full ground truth dataset for the GNN. Following usual practices, this dataset was again split 80% for training and 20% for testing the model.

For the 3D FFN, as mentioned in Section 4.2, E3FP fingerprints are the appropriate input molecular representation type. SMILES string representations of the combinations were converted into E3FP fingerprints using the E3FP Python package [41]. However, the generation of E3FP fingerprints requires significantly more computational power and time than the generation of Morgan Fingerprints or molecular graphs. For this reason, generating E3FP fingerprint representations from the SMILES string representations of the full combinations proved computationally infeasible. Instead, E3FP fingerprints were generated for each half of each combination separately (i.e. the adjuvant and the antibiotic separately) and then concatenated to generate the final E3FP representations of the full combinations. Given that 2048 bits was identified to be the optimal Morgan Fingerprint length for the 2D FFN in Section 4.3.1, 2048-bit E3FPs were generated for each half of the combination and concatenated to create a 4096-bit fingerprint for the full combination. Additionally, given that many molecules can fold into multiple thermodynamically favorable 3-D conformations, each combination had multiple associated E3FP fingerprints. These E3FP fingerprints, along with the 200 RDKit global molecular features, result in input vectors of length 4296 (4096+200) for each combination. These inputs along with the binary synergy scores comprise the full ground truth dataset for the 3D GNN. This dataset was again split 80% for training and 20% for testing the model.

5.3 Machine Learning Model Build

After finalizing the designs for all three types of models, each model was built sequentially as per these designs.

5.3.1 2D FFN

As per the design decisions made for the 2D FFN, as described in Section 4.3.1, the final 2D FFN model architecture was built. The 2D FFN architecture consists of first an input layer (Figure 10). Following the design decision to use 2048-bit Morgan Fingerprints along with 200 RDKit global molecular features as inputs to the model, this input layer has 2048 + 200 = 2248 nodes (Figure 10). Following this input layer are two hidden layers, each with 2048 nodes. Each hidden layer uses the rectified linear unit (ReLU) activation function, which is one of the most commonly-used activation functions since it is more efficient than other activation functions [46]. Following the two hidden layers is a final output layer with a single node. As per the decision described in Section 4.3.1, this output layer has the sigmoid activation function.

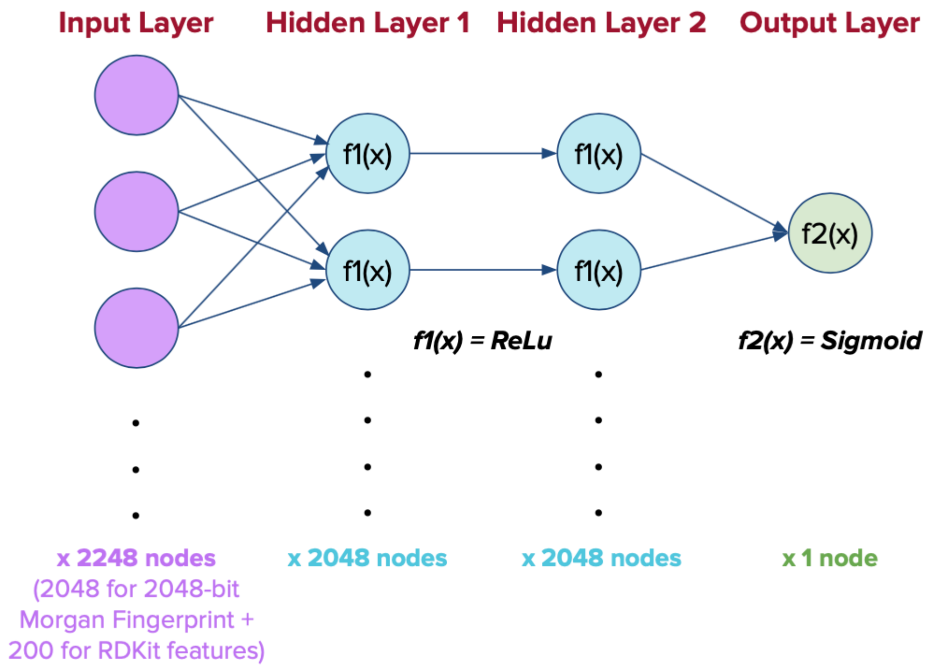


Figure 10 - Schematic of the Final 2D FFN Model Build. The optimal 2D FFN has an input layer with 2248 nodes, 2 hidden layers with 2048 nodes each and the rectified linear unit activation function, and an output layer with one node and the sigmoid activation function.

The training and testing datasets created for this 2D FFN (described in Section 5.2) were input into this model architecture and used for training and testing the model. While training and testing the model, a learning rate of 0.0001 and a dropout rate of 0.1 were used as per the hyperparameter optimization-informed design decisions described in Section 4.3.1. To obtain more robust results while training and testing this 2D FFN, 10 independent ensemble models (essentially 10 independent runs) were trained and tested, resulting in a final sample size of n = 10. Each of these ensemble models was trained and tested for a maximum of 30 epochs each to grant each model sufficient iterations to learn to make predictions. However, an early stopping function was also implemented that ensured that if each model's loss stabilized earlier than 30 epochs, training was stopped to avoid overfitting.

5.3.2 GNN

As per the design decisions described in Section 4.3.2, the final GNN model architecture was built. The architecture consists of first a graph input layer (Figure 11), where the graph representations of the combinations are input. Since the graph representations have 75 elements, there are 75 nodes in this graph input layer (Figure 11). Following this graph input layer are four graph convolutional layers, with sequentially decreasing numbers of nodes and the ReLU activation function. Following the graph convolutional layers, the 200 RDKit global molecular features are input as additional features. Thus, the subsequent layer has 264 nodes (64 nodes for the output from the last graph convolutional layer plus 200 more nodes for the 200 RDKit features). Following this layer are two hidden layers with 200 and 64 nodes respectively and the randomized leaky rectified linear unit (RRelU) activation function. Following the two hidden layers is a final output layer with two nodes and the log_softmax activation function.

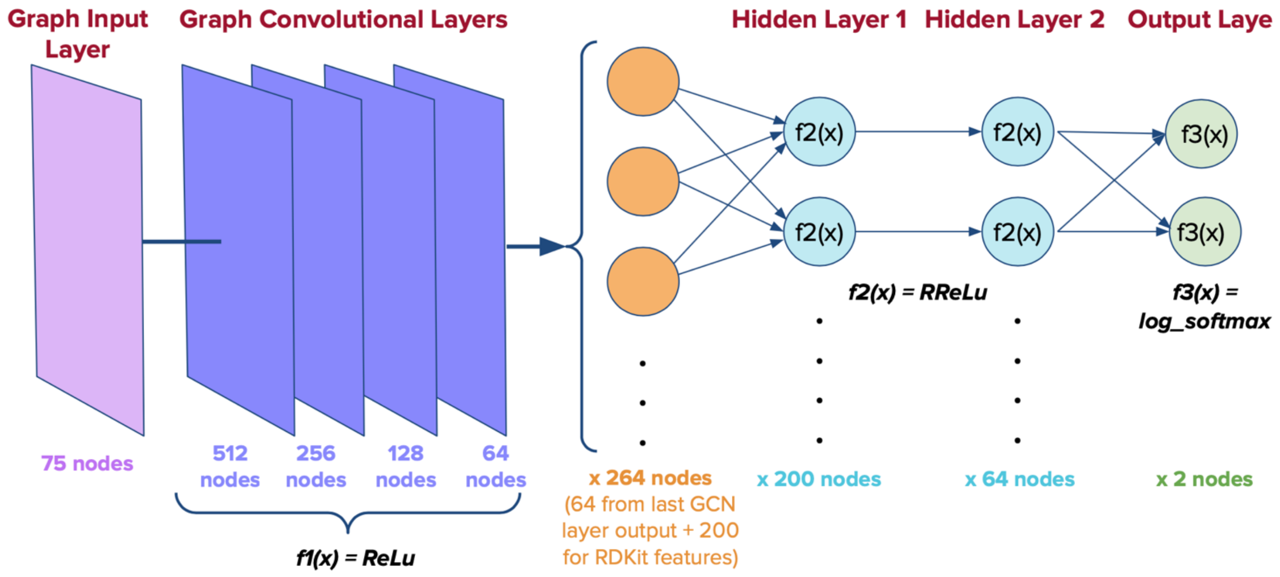


Figure 11 - Schematic of the Final GNN Model Build. The optimal GNN has an input layer with 75 nodes, 4 graph convolutional layers with sequentially decreasing numbers of nodes and the rectified linear unit activation function, 2 hidden layers with 200 and 64 nodes and the random leaky rectified linear unit activation function, and an output layer with 2 nodes and the log_softmax activation function.

The training and testing datasets created for this GNN (described in Section 5.2) were input into this model architecture and used for training and testing the model. A learning rate of 0.0001 and a dropout rate of 0.1 were used as per the hyperparameter optimization-informed design decisions described in Section 4.3.2. Once again, to obtain more robust results while training and testing this GNN, 10 independent ensemble models were trained and tested, resulting in a final sample size of n = 10.

Each of these ensemble models was trained and tested for 20 epochs each to grant each model sufficient iterations to learn to make predictions. In the case of the GNN, the early stopping function that was used with the 2D FFN could not effectively be used. This was because when training the GNN, this early stopping function was observed to halt the training when the loss function had reached a local minimum rather than the global minimum. Thus, to manually identify a point at which training should be halted before overfitting occurred, the progression of the model's loss metric over the epochs was analyzed. It was observed that the loss and performance metrics stabilized by 20 epochs, so 20 epochs were used for training and testing the GNN.

5.3.3 3D FFN

As per the design decisions described in Section 4.3.3, the final 3D FFN model architecture was built. The architecture consists of first an input layer (Figure 12). Following the design decision to use 2048-bit E3FPs for each individual molecule (so 4096 bits for each combination as described in Section 5.2) along with 200 RDKit global molecular features as inputs to the model, this input layer has 4096 + 200 = 4296 nodes (Figure 12). Like in the 2D FFN, following this input layer are two hidden layers, each with 2048 nodes and the rectified linear unit (ReLU) activation function, and then a final output layer with a single node and the sigmoid activation function.

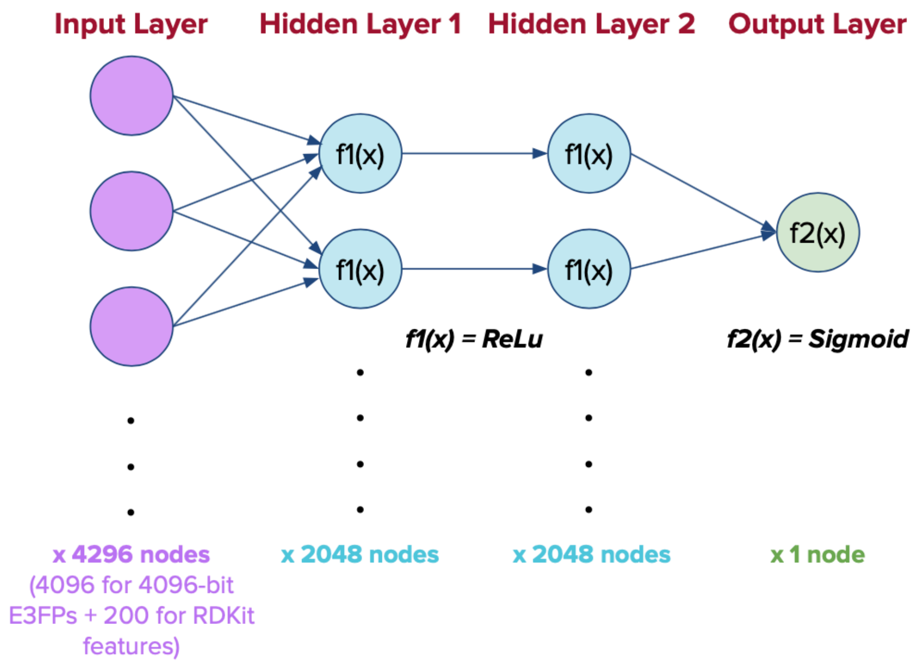
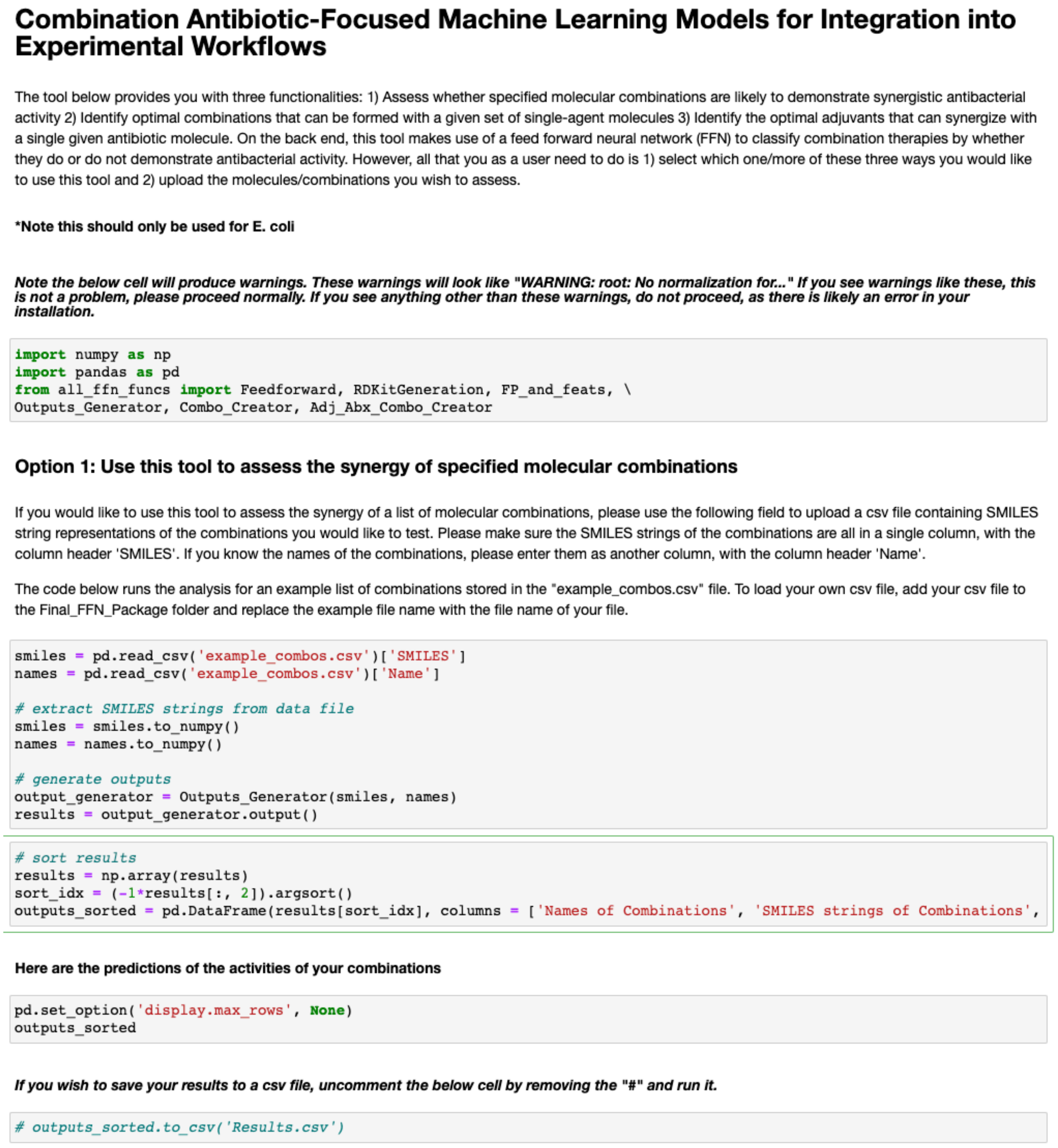


Figure 12 - Schematic of the Final 3D FFN Model Build. The optimal 3D FFN has an input layer with 4296 nodes, 2 hidden layers with 2048 nodes each and the rectified linear unit activation function, and an output layer with 1 node1 and the sigmoid activation function.

The training and testing datasets created for this 3D FFN (described in Section 5.2) were input into this model architecture and used for training and testing the model. While training and testing the model, a learning rate of 0.0001 and a dropout rate of 0.1 were used as per the hyperparameter optimization-informed design decisions described in Section 4.3.3. To obtain more robust results while training and testing this 3D FFN, 10 independent ensemble models were again trained and tested, resulting in a final sample size of n = 10. Each of these ensemble models was again trained and tested for a maximum of 30 epochs each to grant each model sufficient iterations to learn to make predictions. Like with the 2D FFN, an early stopping function was also implemented that ensured that if each model's loss stabilized earlier than 30 epochs, training was stopped to avoid overfitting.

5.4 User Interface Build

A Jupyter Notebook user interface was built as per the necessary design features that had been previously identified (Section 4.4) (Figures 13-15). The completed best-performing final model (identified in 6.3) was deployed in the backend of this Jupyter Notebook user interface.


Combination Antibiotic-Focused Machine Learning Models for Integration into Experimental Workflows

The tool below provides you with three functionalities: 1) Assess whether specified molecular combinations are likely to demonstrate synergistic antibacterial activity 2) Identify optimal combinations that can be formed with a given set of single-agent molecules 3) Identify the optimal adjuvants that can synergize with a single given antibiotic molecule. On the back end, this tool makes use of a feed forward neural network (FFN) to classify combination therapies by whether they do or do not demonstrate antibacterial activity. However, all that you as a user need to do is 1) select which one/more of these three ways you would like to use this tool and 2) upload the molecules/combinations you wish to assess.

***Note this should only be used for E. coli**

Note the below cell will produce warnings. These warnings will look like "WARNING: root: No normalization for..." If you see warnings like these, this is not a problem, please proceed normally. If you see anything other than these warnings, do not proceed, as there is likely an error in your installation.

```
import numpy as np
import pandas as pd
from all_ffn_funcs import Feedforward, RDKitGeneration, FP_and_feats, \
Outputs_Generator, Combo_Creator, Adj_Abx_Combo_Creator
```

Option 1: Use this tool to assess the synergy of specified molecular combinations

If you would like to use this tool to assess the synergy of a list of molecular combinations, please use the following field to upload a csv file containing SMILES string representations of the combinations you would like to test. Please make sure the SMILES strings of the combinations are all in a single column, with the column header 'SMILES'. If you know the names of the combinations, please enter them as another column, with the column header 'Name'.

The code below runs the analysis for an example list of combinations stored in the "example_combos.csv" file. To load your own csv file, add your csv file to the Final_FFN_Package folder and replace the example file name with the file name of your file.

```
smiles = pd.read_csv('example_combos.csv')['SMILES']
names = pd.read_csv('example_combos.csv')['Name']

# extract SMILES strings from data file
smiles = smiles.to_numpy()
names = names.to_numpy()

# generate outputs
output_generator = Outputs_Generator(smiles, names)
results = output_generator.output()
```

```
# sort results
results = np.array(results)
sort_idx = (-1*results[:, 2]).argsort()
outputs_sorted = pd.DataFrame(results[sort_idx], columns = ['Names of Combinations', 'SMILES strings of Combinations',
```

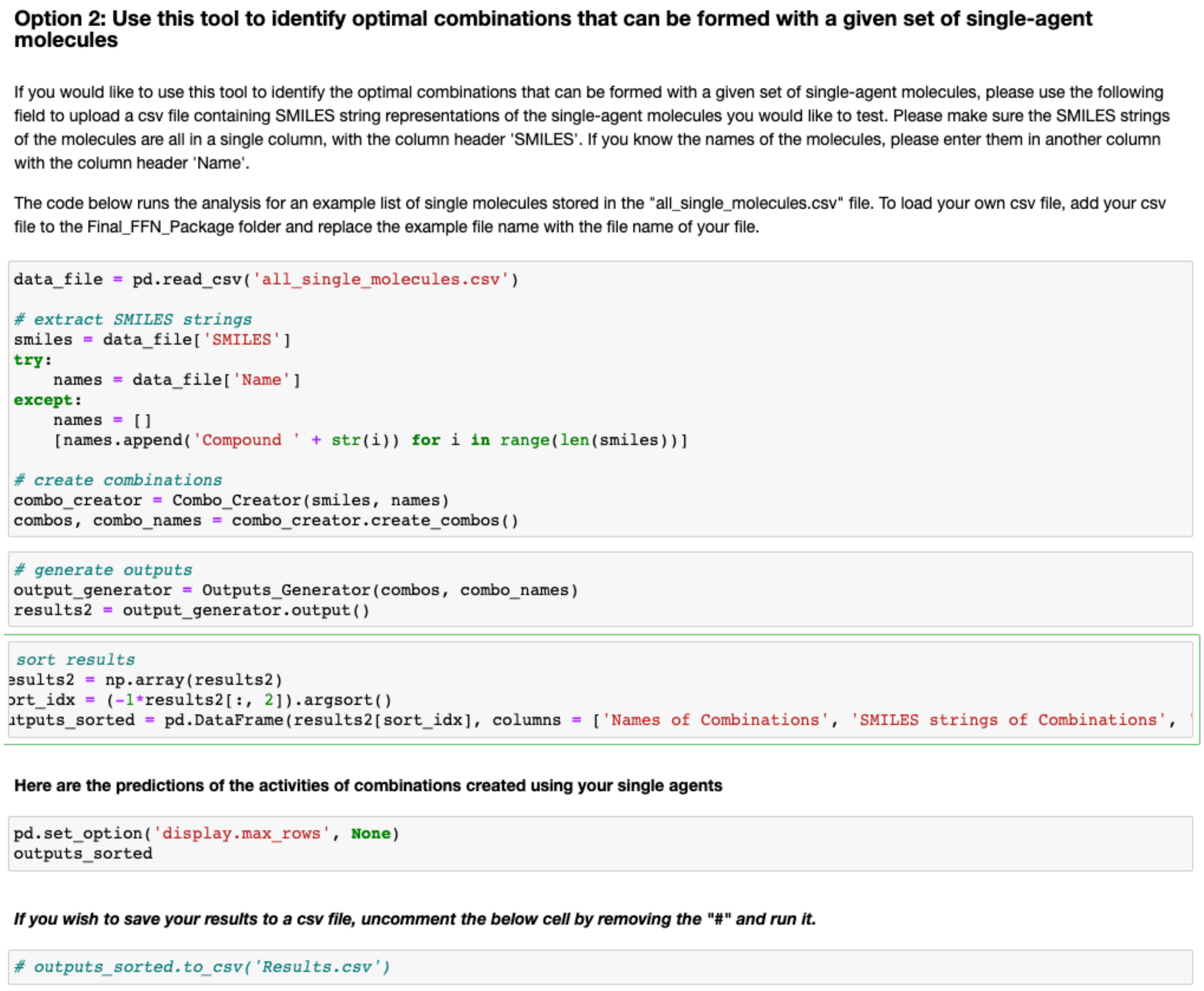
Here are the predictions of the activities of your combinations

```
pd.set_option('display.max_rows', None)
outputs_sorted
```

If you wish to save your results to a csv file, uncomment the below cell by removing the "#" and run it.

```
# outputs_sorted.to_csv('Results.csv')
```

Figure 13 - Jupyter Notebook User Interface: Task 1. This portion of the Jupyter Notebook interface allows the user to assess the synergistic activity of specified molecular combinations.


Option 2: Use this tool to identify optimal combinations that can be formed with a given set of single-agent molecules

If you would like to use this tool to identify the optimal combinations that can be formed with a given set of single-agent molecules, please use the following field to upload a csv file containing SMILES string representations of the single-agent molecules you would like to test. Please make sure the SMILES strings of the molecules are all in a single column, with the column header 'SMILES'. If you know the names of the molecules, please enter them in another column with the column header 'Name'.

The code below runs the analysis for an example list of single molecules stored in the "all_single_molecules.csv" file. To load your own csv file, add your csv file to the Final_FFN_Package folder and replace the example file name with the file name of your file.

```
data_file = pd.read_csv('all_single_molecules.csv')

# extract SMILES strings
smiles = data_file['SMILES']
try:
    names = data_file['Name']
except:
    names = []
    [names.append('Compound ' + str(i)) for i in range(len(smiles))]

# create combinations
combo_creator = Combo_Creator(smiles, names)
combos, combo_names = combo_creator.create_combos()
```

```
# generate outputs
output_generator = Outputs_Generator(combos, combo_names)
results2 = output_generator.output()
```

```
 sort results
esults2 = np.array(results2)
ort_idx = (-1*results2[:, 2]).argsort()
utputs_sorted = pd.DataFrame(results2[sort_idx], columns = ['Names of Combinations', 'SMILES strings of Combinations', 
```

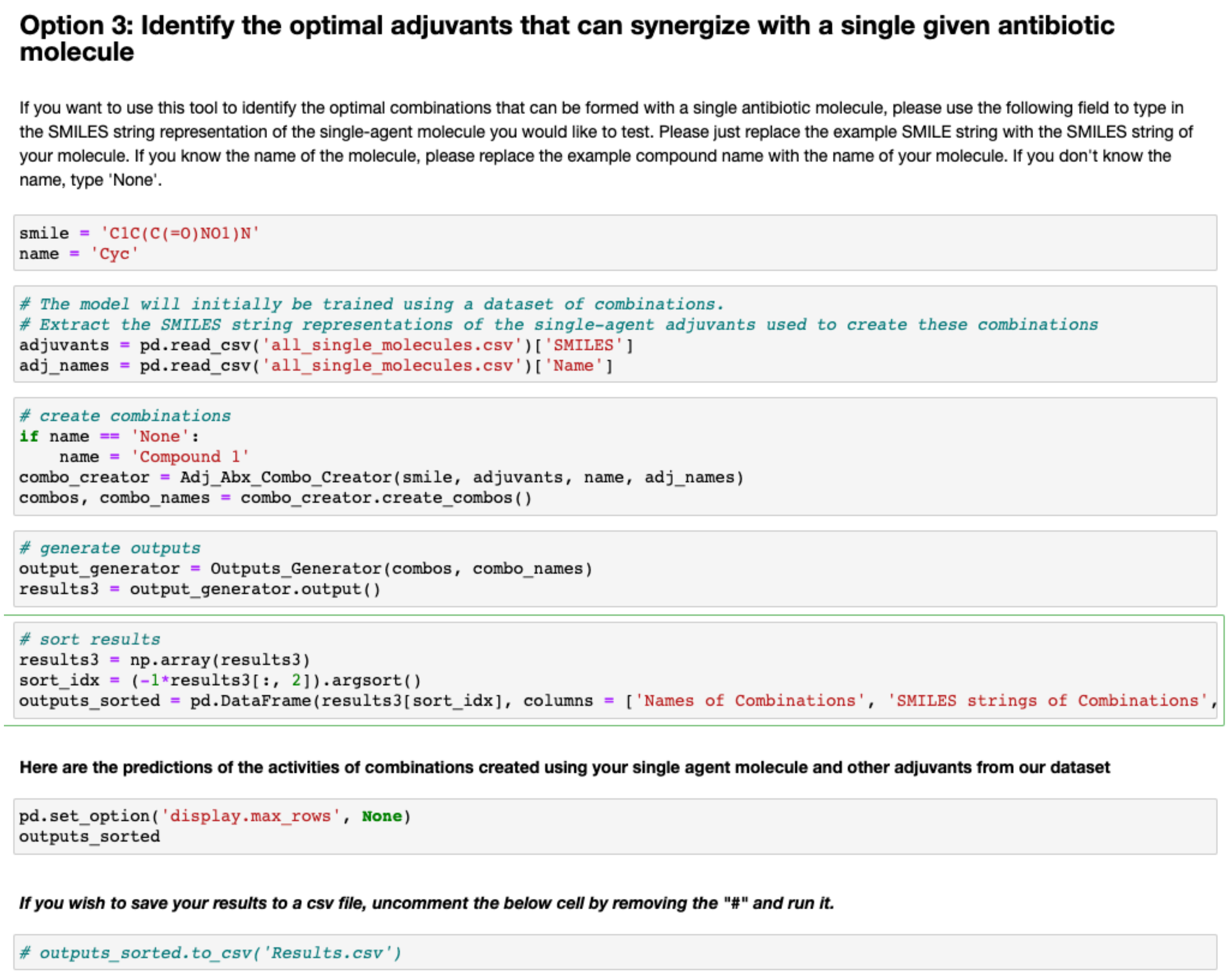
Here are the predictions of the activities of combinations created using your single agents

```
pd.set_option('display.max_rows', None)
outputs_sorted
```

If you wish to save your results to a csv file, uncomment the below cell by removing the "#" and run it.

```
# outputs_sorted.to_csv('Results.csv')
```

Figure 14 - Jupyter Notebook User Interface: Task 2. This portion of the Jupyter Notebook interface allows the user to identify the optimal combinations that can be formed with a given set of single-agent molecules.


Option 3: Identify the optimal adjuvants that can synergize with a single given antibiotic molecule

If you want to use this tool to identify the optimal combinations that can be formed with a single antibiotic molecule, please use the following field to type in the SMILES string representation of the single-agent molecule you would like to test. Please just replace the example SMILE string with the SMILES string of your molecule. If you know the name of the molecule, please replace the example compound name with the name of your molecule. If you don't know the name, type 'None'.

```
smile = 'C1C(C(=O)NO1)N'
name = 'Cyc'
```

```
# The model will initially be trained using a dataset of combinations.
# Extract the SMILES string representations of the single-agent adjuvants used to create these combinations
adjuvants = pd.read_csv('all_single_molecules.csv')['SMILES']
adj_names = pd.read_csv('all_single_molecules.csv')['Name']
```

```
# create combinations
if name == 'None':
    name = 'Compound 1'
combo_creator = Adj_Abx_Combo_Creator(smile, adjuvants, name, adj_names)
combos, combo_names = combo_creator.create_combos()
```

```
# generate outputs
output_generator = Outputs_Generator(combos, combo_names)
results3 = output_generator.output()
```

```
# sort results
results3 = np.array(results3)
sort_idx = (-1*results3[:, 2]).argsort()
outputs_sorted = pd.DataFrame(results3[sort_idx], columns = ['Names of Combinations', 'SMILES strings of Combinations',
```

Here are the predictions of the activities of combinations created using your single agent molecule and other adjuvants from our dataset

```
pd.set_option('display.max_rows', None)
outputs_sorted
```

If you wish to save your results to a csv file, uncomment the below cell by removing the "#" and run it.

```
# outputs_sorted.to_csv('Results.csv')
```

Figure 15 - Jupyter Notebook User Interface: Task 3. This portion of the Jupyter Notebook interface allows the user to identify the optimal adjuvants that can synergize with a single given antibiotic molecule.

As shown, this user interface allows the user to perform three key tasks, as per the necessary design features described in Section 4.4 (Figures 13-15). For each of these tasks, the interface requires no more information from the user than the file path for their dataset (for Tasks 1 and 2) or the SMILES string representation of the antibiotic molecule being tested (for Task 3). Outputs are returned to the user in an easily viewable table, and the user is also given the option to save the outputs as a CSV file in their local computer for further analysis.

6 Evaluation/Verification

6.1 Measuring Accuracy

In order to assess the utility of the predictions made by the model to an experimentalist, model accuracies were evaluated with auROC, auPR, FPR, and FNR.

6.1.1 auROC

In order to calculate model auROCs, ROC curves were first generated using the roc_curve function in the Python Scikit-learn package [36]. This function first creates a range of thresholds. Then, for each of these thresholds, the FPR and TPR are calculated, e.g. for a given threshold, the FPR corresponding to that threshold is the FPR of predictions with a predicted score ≥ threshold and the TPR corresponding to that threshold is the TPR of predictions with a predicted score ≥ threshold [36].

The FPR and TPR values for all the thresholds were output by the roc_curve function. These FPR and TPR values were then plotted against each other, resulting in a final ROC plot. Thus, using the roc_curve function, ROC curves were generated for all three model classes (Figure 16). Additionally, the ROC curve for a completely random classifier was also plotted for comparison (Figure 16). Qualitative observation reveals that the ROC curve of the 2D FFN is indicative of better performance than those of the GNN and 3D GNN, as the 2D FFN ROC curve is much farther to the upper left of the random classifier ROC curve (indicative of high TPR and low FPR) (Figure 16).

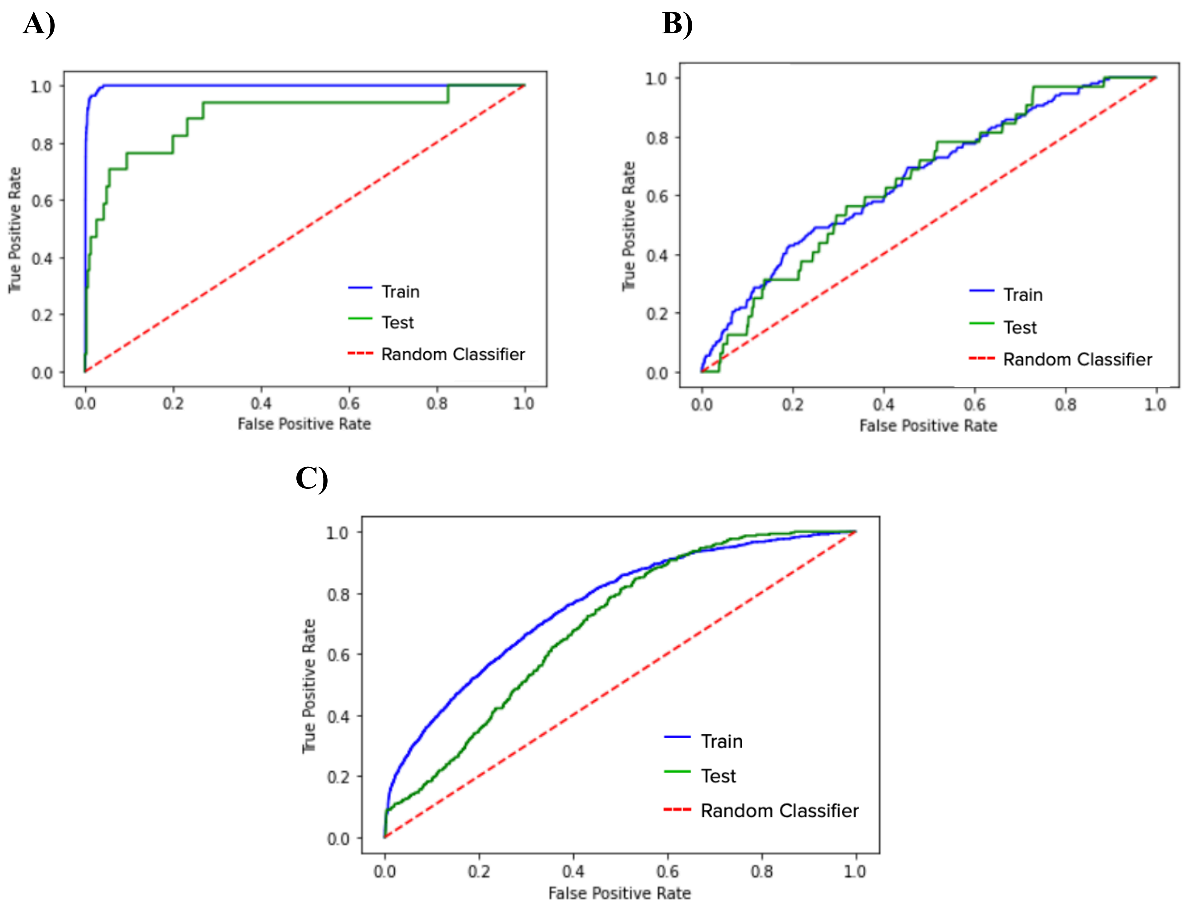


Figure 16 – ROC Curves of the Three Models. The plots of the ROC curves of the **A)** 2D FFN, **B)** GNN, and **C)** 3D FFN demonstrate differences in performance among the three models. Optimal ROC curves demonstrate high TPR and low FPR, so they curve as far to the upper left as possible. Thus, out of these three models, the 2D FFN model has an ROC curve indicating the best performance.

After the generation of these curves, the auROC was calculated by simply calculating the area under these curves once again using the Scikit-learn package [36]. This was repeated for all 10 ensemble models for each model class, and the final auROC metrics of the 10 ensemble models of each model class were compared (Figure 17). The one-tailed Mann-Whitney U significance test was also performed to determine whether differences in auROCs between different model classes were significant (Figure 17).

These comparisons revealed that the 2D FFN demonstrates significantly greater auROCs compared to the GNN and the 3D FFN (Figure 17). These results are consistent in both the training set and the testing set. Thus, these quantitative observations confirm the qualitative observations when comparing the ROC curves of the three model classes (Figure 16).

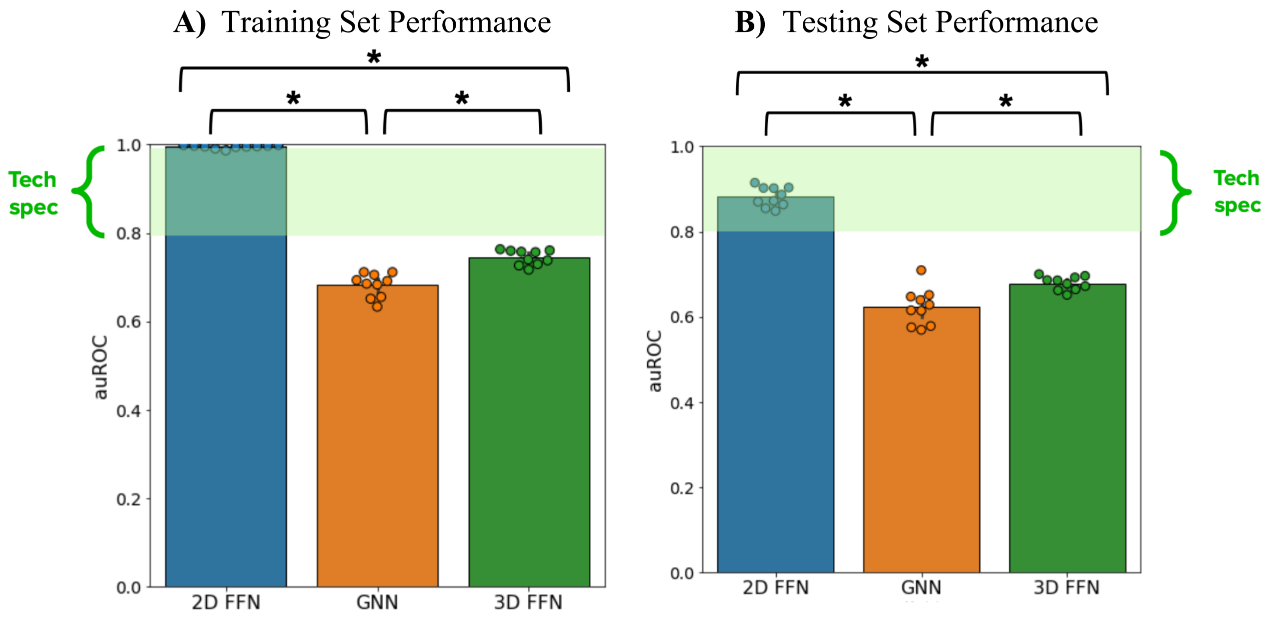


Figure 17 – auROC Measurements of the Three Model Types. A) The auROC metrics of the ensemble models of each of the three model types show differences in performance among the three models when run on the training set (n=10). The 2D FFN average train set auROC (0.99) is significantly greater than the GNN average train set auROC (0.68) and the 3D FFN average train set auROC (0.74). **B)** The auROC metrics of the ensemble models of each of the three model types show differences in performance among the three models when run on the testing set (n=10). The 2D FFN average test set auROC (0.88) is significantly greater than the GNN average test set auROC (0.62) and the 3D FFN average test set auROC (0.68). *(* p < 0.05, one-tailed Mann-Whitney U Test used for all statistical comparisons).*

6.1.2 auPR

In order to calculate model auPRs, PR curves were first generated using the precision_recall_curve function in the Python Scikit-learn package [36]. This function again creates a range of thresholds. Then, for each of these thresholds, the precision and recall are calculated, e.g. for a given threshold, the precision corresponding to that threshold is the precision of predictions with a predicted score ≥ threshold and the recall corresponding to that threshold is the recall of predictions with a predicted score ≥ threshold [36].

Precision and recall values for all the thresholds were output by the precision_recall_curve function and were plotted against each other, resulting in final PR plots for all three model classes (Figure 16). Qualitative observation reveals that the PR curve of the 2D FFN is once again indicative of better performance than those of the GNN and 3D GNN, as the 2D FFN PR curve is much farther to the upper right (indicative of high precision and high recall) (Figure 18).

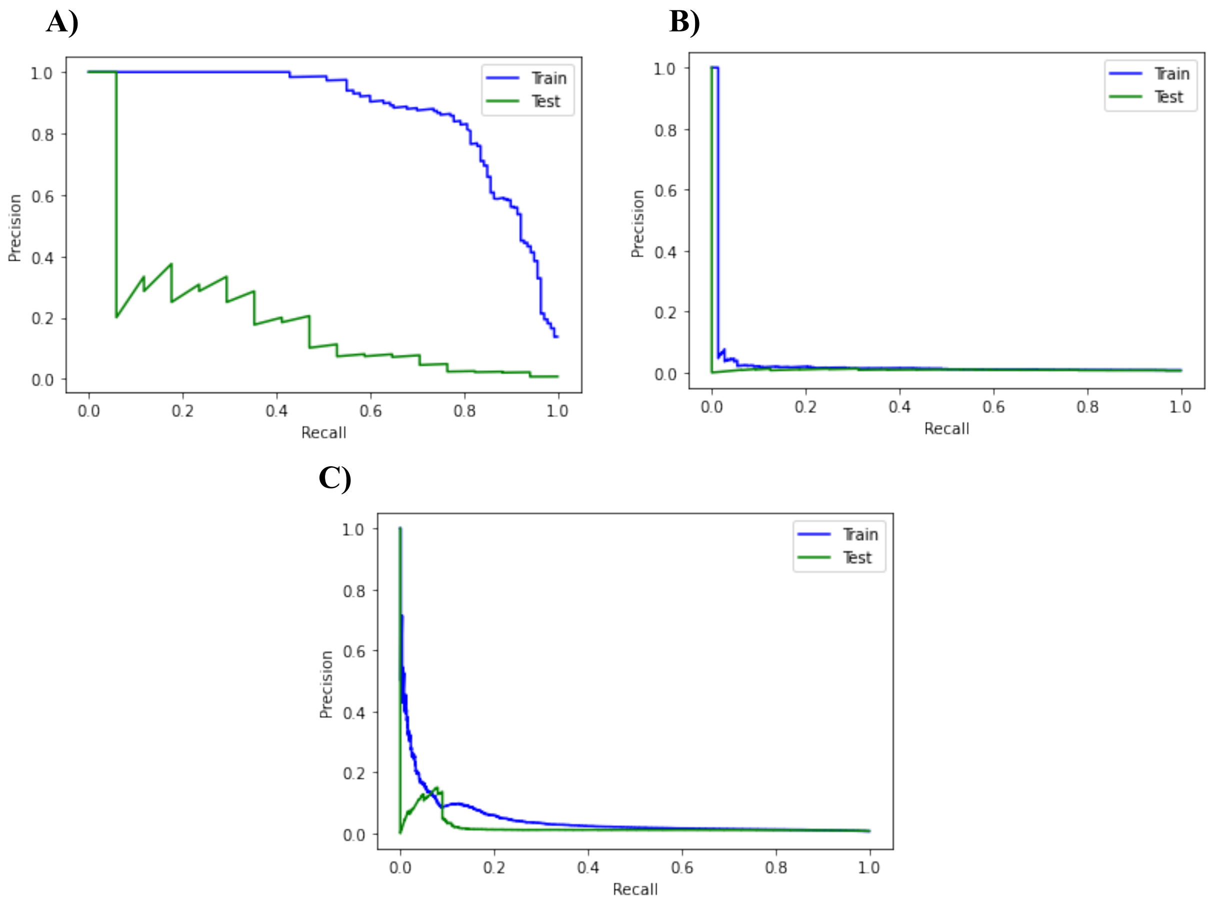


Figure 18 – PR Curves of the Three Models. The plots of the PR curves of the **A)** 2D FFN, **B)** GNN, and **C)** 3D FFN demonstrate differences in performance among the three models. Optimal PR curves demonstrate high recall and high precision, so they curve as far to the upper right as possible. Thus, out of these three models, the 2D FFN model has a PR curve indicating the best performance.

After the generation of these curves, the auPR was calculated by simply calculating the area under these curves once again using the Scikit-learn package [36]. This was repeated for all 10 ensemble models for each model class, and the final auPR metrics of the 10 ensemble models of each model class were compared (Figure 17). The one-tailed Mann-Whitney U significance test was also again performed to determine whether differences in auPRs between different model classes were significant (Figure 19).

These comparisons revealed that the 2D FFN demonstrates significantly greater auPRs compared to the GNN and the 3D FFN in both the training set and the testing set (Figure 19). Thus, these quantitative observations again confirm the qualitative observations when comparing the PR curves of the three model classes (Figure 18).

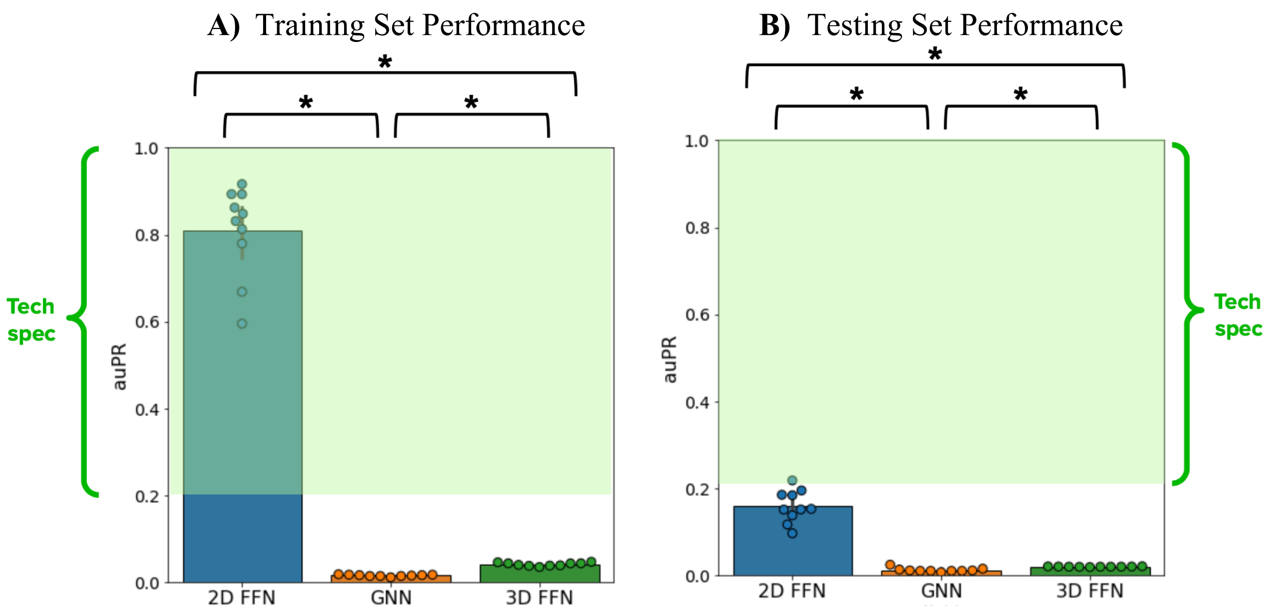


Figure 19 – auPR Measurements of the Three Model Types. A) The auPR metrics of the ensemble models of each of the three model types show differences in performance among the three models when run on the training set (n=10). The 2D FFN average train set auPR (0.81) is significantly greater than the GNN average train set auPR (0.015) and the 3D FFN average train set auPR (0.041). **B)** The auPR metrics of the ensemble models of each of the three model types show differences in performance among the three models when run on the testing set (n=10). The 2D FFN average test set auPR (0.16) is significantly greater than the GNN average test set auPR (0.012) and the 3D FFN average test set auPR (0.020). *(* $p < 0.05$, one-tailed Mann-Whitney U test used for all statistical comparisons).*

6.1.3 FPR

In order to calculate model FPRs, the number of false positives (FP) and true negatives (TN) were determined using the confusion_matrix function in the Python Scikit-learn package [36]. Then, using Equation 2, the FPR was calculated using the FP and TN values. This was repeated for all 10 ensemble models for each model class, and the final FPR metrics of the 10 ensemble models of each model class were compared (Figure 18). The one-tailed Mann-Whitney U significance test was also again performed to determine whether differences in FPRs between different model classes were significant (Figure 20).

These comparisons reveal that the 3D FFN demonstrates the lowest FPRs out of the three model types in both the training set and the testing set, closely followed by the 2D FFN (Figure 20). Differences between the FPRs of the three models are significant in the training set (Figure 20). In the testing set, differences between the 2D FFN and GNN and differences between the GNN and 3D FFN are significant, but the differences between the 3D FFN and 2D FFN are not.

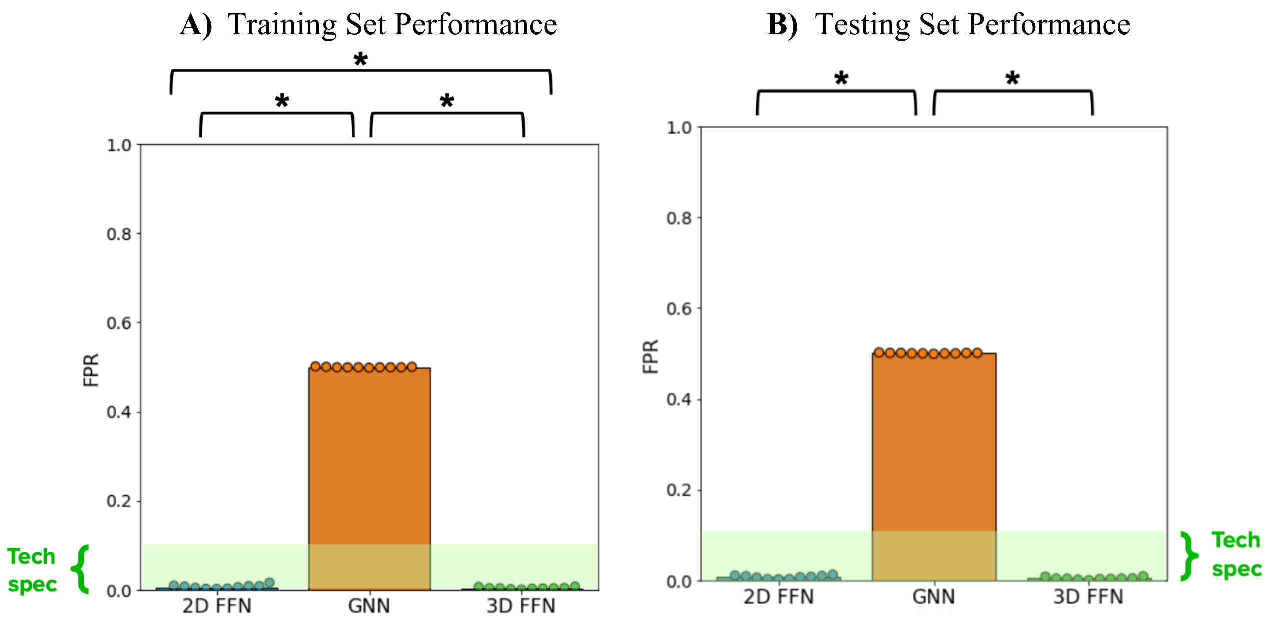


Figure 20 – FPR Measurements of the Three Model Types. A) The FPR metrics of the ensemble models of each of the three model types show differences in performance among the three models when run on the training set (n=10). The 3D FFN average train set FPR (0.0031) is significantly lower than the 2D FFN model average train set FPR (0.0062) and the GNN model average train set FPR (0.50). **B)** The FPR metrics of the ensemble models of each of the three model types show differences in performance among the three models when run on the testing set (n=10). The 3D FFN average test set FPR (0.0036) is significantly lower than the GNN average test set FPR (0.50) but is not significantly different from the 2D FFN average test set FPR (0.0066). *(* $p < 0.05$, one-tailed Mann-Whitney U test used for all statistical comparisons).*

6.1.4 FNR

In order to calculate model FNRs, the number of false negatives (FN) and true positives (TP) were determined again using the confusion_matrix function in the Python Scikit-learn package [36]. Then, using Equation 3, the FNR was calculated using the FN and TP values. This was repeated for all 10 ensemble models for each model class, and the final FNR metrics of the 10 ensemble models of each model class were compared (Figure 19). The one-tailed Mann-Whitney U significance test was also again performed to determine whether differences in FNRs between different model classes were significant (Figure 21).

These comparisons reveal that in the training set, the 2D FFN demonstrates the lowest FNRs out of the three model types, followed by the GNN (Figure 21). In the testing set, the GNN demonstrates the lowest FNRs out of the three model types, followed by the 2D FFN (Figure 21). Differences between the FNRs of the three model types are significant in both the training and testing sets (Figure 21).

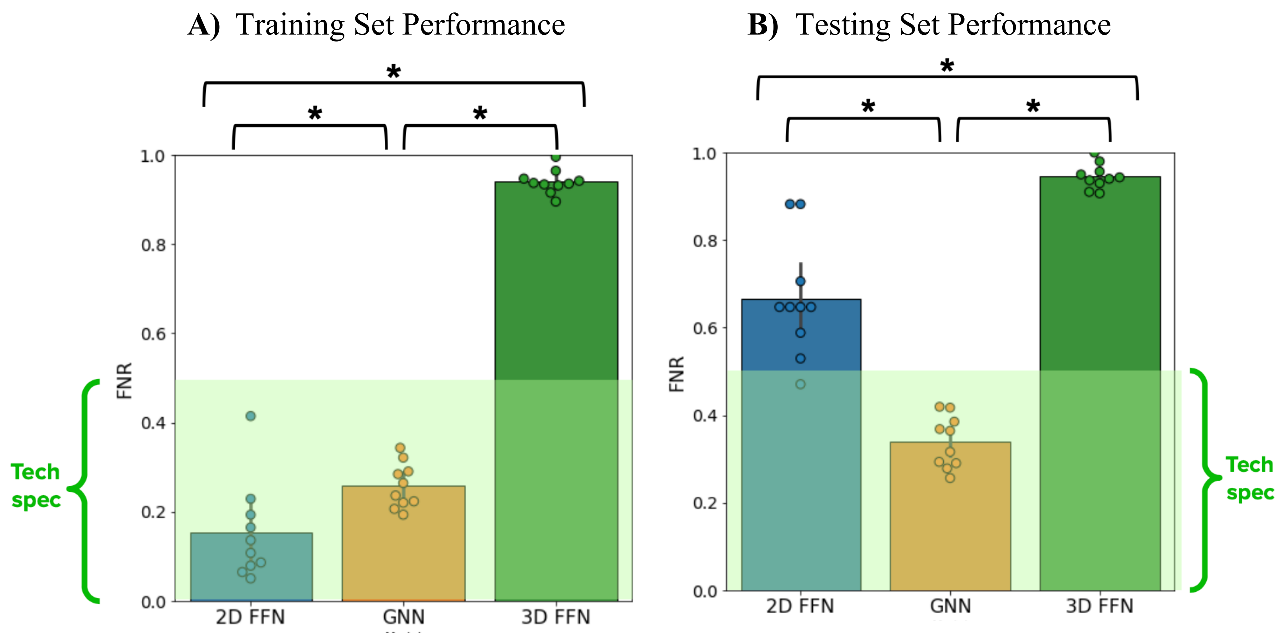


Figure 21 – FNR Measurements of the Three Model Types. A) The FNR metrics of the ensemble models of each of the three model types show differences in performance among the three models when run on the training set (n=10). The 2D FFN average train set FNR (0.15) is significantly lower than the GNN average train set FNR (0.26) and the 3D FFN average train set FNR (0.94). **B)** The auPR metrics of the ensemble models of each of the three model types show differences in performance among the three models when run on the testing set (n=10). The GNN average test set FNR (0.34) is significantly lower than the 2D FFN average test set FNR (0.66) and the 3D FFN average test set FNR (0.95). *(* $p < 0.05$, one-tailed Mann-Whitney U test used for all statistical comparisons).*

6.1.5 Selection of Best-Performing Model and Final Model Assembly

To identify the best-performing model, the average performances of each of the three model types with regards to the four accuracy-related technical specifications were compared (Table 6).

Table 6. Comparison of the performances of the three model types to the accuracy technical specifications

	Technical Specification	2D FFN		GNN		3D FFN	
		Train	Test	Train	Test	Train	Test
Average auROC	≥ 0.8	0.99 ✓	0.88 ✓	0.68 X	0.62 X	0.74 X	0.68 X
Average auPR	≥ 0.2	0.81 ✓	0.16 X	0.015 X	0.012 X	0.041 X	0.020 X
Average FPR	≤ 0.1	0.0062 ✓	0.0066 ✓	0.50 X	0.50 X	0.0031 ✓	0.0036 ✓
Average FNR	≤ 0.5	0.15 ✓	0.66 X	0.26 ✓	0.34 ✓	0.94 X	0.95 X

Broadly, the 2D FFN meets the greatest proportion of the accuracy-related technical specifications (Table 6). In fact, the 2D FFN meets almost all these accuracy specifications. Even when considering the specifications that it does not meet, the margin between the 2D FFN average measured metrics and these specifications are not extremely wide. In contrast, the GNN falls short of most of the accuracy specifications, although it does demonstrate the best FNRs out of the three model types. Likewise, the 3D FFN also misses many of the specifications, although it demonstrates the best FPRs out of the three model types.

Therefore, when broadly considering all the accuracy specifications, the 2D FFN is the clear choice for the best model type with which to proceed. After selecting the model type, the next step was to prepare a final assembled model for all further testing. As mentioned before, all three model classes were trained and tested with 10 independent ensemble models. To prepare this final assembled model, the 10 ensemble models of the 2D FFN were combined by averaging their predictions for each input, which aids in improving the robustness and reliability of the output that the user receives.

6.2 Measuring Computational Intensiveness

In order to assess the feasibility of the use of the model on a computer with specifications less than or equal to a standard experimentalist's computer, the computational intensiveness of the final model (prepared as described in 6.1.5) was assessed by measuring model speed, RAM usage, storage consumption, and the largest input file size allowed by the model. Due to accessibility constraints, these tests were performed on a CPU with 8 GB RAM and 256 GB storage, so the CPU on which the model was tested had lower RAM but equal storage compared to the CPUs of Collins Lab experimentalists. This means that if the model were to be run on an experimentalist's CPU, the computational efficiency would be either equal or better than the measurements described below.

6.2.1 Model Speed

To measure model speed, a random sample of 5% of the combinations in the full ground-truth dataset (amounting to 1364 combinations) was created and input into the final 10 ensembles-combined model. Using a simple timing function in Python, the elapsed time between the point when this sample was input into the model and the point when the final outputs to the user were calculated was measured. The total model speed was then calculated by dividing 1364 combinations by these elapsed times, as per Equation 6. However, this is the speed of running all 10 of the ensemble models, so the elapsed time is over-inflated since it is the time to run 10 ensemble models sequentially. To correct for this and obtain the relevant parameter, the speed of an individual ensemble model, the speed was multiplied by 10:

$$\boldsymbol{Model\ Speed} = \frac{\boldsymbol{1364\ combinations}}{\frac{\boldsymbol{Elapsed\ time\ (sec)}}{\boldsymbol{10}}} = \boldsymbol{10\ x\ \frac{1364\ combinations}{Elapsed\ time\ (sec)}} \qquad (7)$$

This process was repeated for 10 random samples of 5% of the combinations in the full ground-truth dataset, for a final sample size of n = 10. The average final speed of the model is 2523.26 $\pm$ 77.21 combinations/second.

6.2.2 RAM Required to Run Model

To measure RAM usage, increasingly larger batches of combinations were input into the final 10 ensembles-combined model. Using built-in functions in the Python psutil package [37], the RAM usage was measured while the model was generating outputs for each batch of combinations. This whole process was then repeated three times. Results indicated that RAM usage initially increased rapidly as the number of combinations in the input data was increased but then plateaued around 2-3 GB (Figure 22).

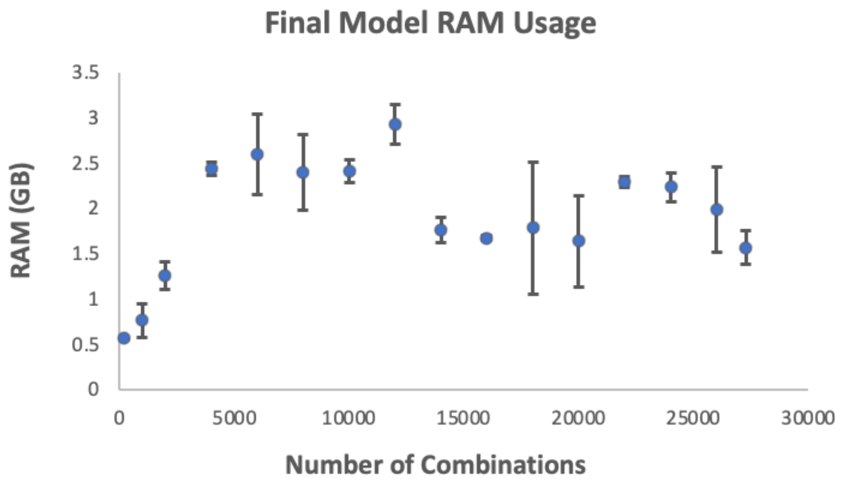


Figure 22 – RAM usage by the final model. The RAM (GB) used by the final model to process and generate predictions for increasingly larger batches of combinations (mean +/- std shown in graph) (n=3) increases steeply initially but then plateaus around 2-3 GB.

Thus, the maximum RAM usage for the final product, irrespective of the number of molecules in the input dataset, is around 2-3 GB.

6.2.3 Storage Required to Download Model

The storage required to download the model was calculated by measuring the following:

1. File size of the Jupyter Notebook user interface
2. Sum of the file sizes of the Python script files that are used by this Jupyter Notebook (and are therefore necessary supporting files to run the user interface)
3. Sum of the sizes of the 10 ensemble models (the sizes of the files needed to load the trained models into the user interface)

4. Size of the folder containing the virtual environment from which the Jupyter Notebook must be run (this virtual environment contains all the packages and dependencies needed to run the Jupyter Notebook and the associated scripts)

These file and folder sizes were measured simply by observing the file/folder size through regular computer functions. Given that these were being measured on a Mac, file and folder sizes were recorded using the default information provided in the Finder application.

The results of these measurements revealed that:

1. Size of the folder containing the following items = 370.5 MB
    a. Jupyter Notebook user interface
    b. Python scripts used by this Jupyter Notebook
    c. 10 ensemble models for the 2D FFN
    d. Example single-molecule and combinations datasets
    e. Installation guide
2. Size of the virtual environment within which the Jupyter Notebook is run = 1.99 GB

The total sum of all these sizes is 2.3605 GB. Thus, the total storage occupied by the final product created in this project is 2.3605 GB.

6.2.4 Largest Input File Size Allowed by Model

To measure the largest input file size, a batch of 100,000 molecules was created using the batch of approximately 250,000 molecules from the ZINC database [47]. When this 100,000-molecule batch was input into the final model, it was able to successfully process and generate predicted outputs for these molecules. The file size of this dataset of 100,000 molecules was 5.1 MB. Next, a batch of 1 million molecules was similarly generated using the molecules from the ZINC database [47]. The file size of this dataset was 52.2 MB. As mentioned in Section 3.2.1.1, Collins Lab experimentalists typically aim to process batches of 1,000-10,000 combinations. Thus, even when increasing the size of the input dataset well beyond the typical batch sizes used by experimentalists, the file size does not come close to the 1 GB threshold (see Section 3.2.2.4).

Thus, although ≥ 1GB was defined as the technical specification for the maximum input file size, the results of these tests showed that a file size limit is not actually encountered when using this model for practical purposes.

6.3 Measuring User-Friendliness

To measure the final product's user-friendliness, the Jupyter Notebook user interface, all 10 trained ensemble models for the 2D FFN, all the required Python scripts, and example datasets were packaged into a single folder.

In addition to these files, it is also necessary to provide a way for any user to clone the virtual environment from which the model is run. The virtual environment contains all the dependencies and Python packages required to run the commands in the Jupyter Notebook user interface. In order to successfully use the final product, each user must locally create the same virtual environment and run the Jupyter Notebook from within this environment. To create a way of doing so, a .yml file was created that contains all the details of the virtual environment necessary for duplicating the environment. To supplement this .yml file, an additional file containing the names of a few additional necessary packages was created.

To aid the user in setting up the environment and running and utilizing the Jupyter Notebook interface, an installation guide with step-by-step instructions was created (Appendix 5). Finally, all of the above folders and files were sent to Collins Lab experimentalists, along with a survey to collect qualitative and quantitative feedback (Appendix 4). Three Collins Lab experimentalists were able to test the final product within the timeline of this project. The product received a user-friendliness rating of 5 from all three experimentalist (Figure 23).

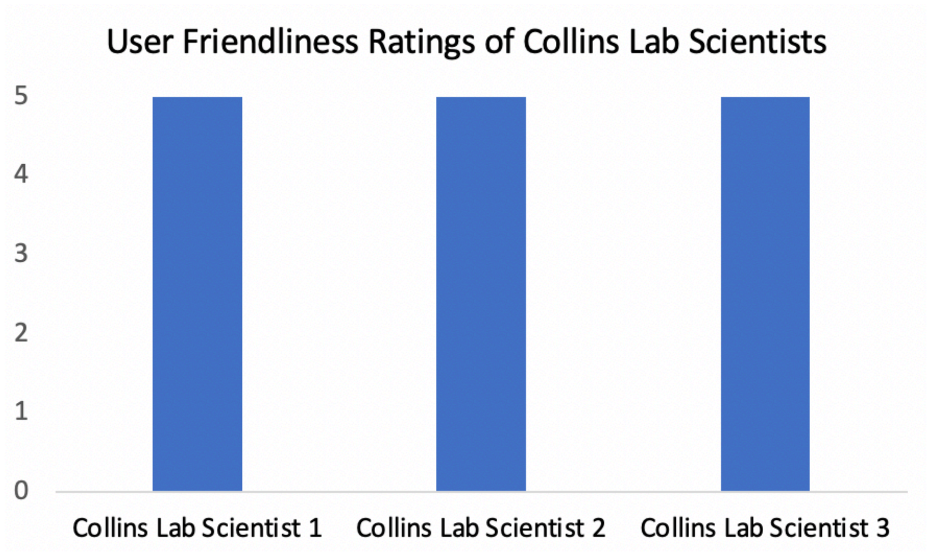


Figure 23 – Evaluation of the User-Friendliness of the User Interface. Quantitative ratings by experimentalists of the user-friendliness of the final product demonstrate a consistent rating of 5 (n=3).

Additionally, qualitative feedback from all three experimentalists indicated very positive experiences while using the product. Specific comments from Collins Lab Scientist 1 included “It was great, and super easy to use for someone with very minimal computation experience!” and “Very easy, instructions in the installation guide and in the jupyter notebook were easy to follow.”

Collins Lab Scientist 2 likewise reported ease and comfort with using the final product and no significant issues. Scientist 2 did report a few errors during the installation process as well as some commands that were run differently by Scientist 2 due to differences in Anaconda versions. Scientist 2 also recommended renaming the optional output files generated by the Jupyter Notebook user interface to avoid overwriting issues. However, overall, both experimentalists reported ease and comfort while using the final product.

Collins Lab Scientist 3 also reported an overall positive experience, with specific comments including: 1) “Yes, easy to use and I look the inclusion of example tests to run to use as a basis for any compound sets that I would want to load myself.” 2) “I would feel comfortable using this tool.” Scientist 3 did, however, report some challenging errors during the installation process, especially during the installation of PyTorch and its dependencies. Scientist 3 also recommended including additional background information within the notebook regarding the training of the model that is deployed in the Jupyter Notebook to provide the user context regarding the model being used.

6.4 Evaluation of Final Product against Technical Specifications

Once the accuracy, computational intensives, and user-friendliness metrics were calculated for the final model and Jupyter Notebook, the metrics were evaluated against the technical specifications (Table 7).

Table 7. Comparison of final accuracy, computational intensiveness, and user-friendliness metrics with technical specifications.

Technical Specification	**Final Product Evaluation**	
Accuracy		
	Training	**Testing**
auROC ≥ 0.8	0.99 ✓	0.88 ✓
auPR ≥ 0.2	0.81 ✓	0.16 **X**
FPR ≤ 0.1	0.0062 ✓	0.0066 ✓
FNR ≤ 0.5	0.15 ✓	0.66 **X**
Computational Intensiveness		
Model Speed ≥ 1200 combos/sec	2523.26 combos/sec ✓	
RAM Usage ≤ 8 GB	Plateaus at 2-3 GB ✓	
Storage Usage ≤ 2.5 GB	2.3605 GB ✓	
Largest Allowed Input File Size ≥ 1 GB	No file size limitation found --	
User-Friendliness		
Qualitative feedback indicative of ease and comfort of use	✓	
Average user-friendliness rating ≥ 4	5 ✓	

As can be seen, almost all the technical specifications have been met by the final product (Table 7). The only two specifications that were not met are the auPR and FNR specifications in the testing set. However, even for these specifications, the measured values are not unexpectedly far from the technical specification. Especially considering that model performance is generally poorer in the testing set than the training set, the auPR and FNR metrics in the testing set, despite being somewhat out of range of the technical specifications, are still of decent quality. While models are not typically evaluated according to their training set performance, the inclusion of these training set metrics in addition to the test set evaluation is a helpful baseline for the model.

Thus, overall, the final product demonstrates extremely positive performance with regards to the technical specifications and goals that were used to guide its development.

7 Conclusions and Future Work

In summary, the final product created in this project is a Jupyter Notebook user interface implementing a 2D FFN that can predict the synergistic antibacterial efficacy of combination antibiotic therapies. Furthermore, this 2D FFN can make these predictions requiring only SMILES string inputs from the user. Thus, this final product can allow a user to analyze datasets of combinations or single-agent molecules without needing to obtain any information beyond simple SMILES string representations of the molecules they wish to analyze.

There are a few possible reasons that may explain why the 2D FFN outperformed the GNN and the 3D GNN despite contrary initial expectations [see Section 4.3]. With regards to the GNN, it is possible that the molecular graph representations that served as inputs for the GNN were of poorer quality than the Morgan Fingerprints that were inputs for the 2D FFN. Morgan Fingerprints are a very well-established type of molecular representation, with a well-tested process for generating them. Additionally, Morgan Fingerprints have the ability to capture a nearly infinite chemical space, as they can incorporate all atoms, hybridizations, etc. In contrast, molecular graph representations in the GNN can sometimes be biased by other data in the training set for the model. Thus, because of these factors, it is possible that the quality of the graph representations in the GNN was poorer than the quality of the Morgan Fingerprints in the 2D FFN. With regards to the 3D FFN, E3FP fingerprints that served as inputs for the 3D FFN are an even newer molecular representation method than the molecular graph representations. Thus, the process of generating them has not been as well tested, and their ability to serve as inputs for these kinds of models has also not been as thoroughly tested as Morgan Fingerprints. Thus, quality of the input could once again have influenced the poorer performance of the 3D FFN. It is also possible that more nodes or hidden layers were needed in the 3D FFN to account for the larger data size of the E3FP fingerprints. However, with memory being a restriction, it was not possible to increase the number of nodes or hidden layers further. It is possible that if the memory restrictions were loosened, it could be possible to further improve the performance of the 3D FFN, but this would require further testing.

As discussed in Section 6.4, almost all of the technical specifications that were initially created have been met by the final product (Table 7). The only two specifications that were not met are the auPR and FNR specifications in the testing set. However, considering that machine learning model performance is generally poorer in the testing set than the training set, the margin between the auPR and FNR measurements and specifications is not unexpectedly large. Therefore, the final product created in this project meets most of the accuracy, computational-intensiveness, and user-friendliness goals and specifications that guided its development.

The final product created in this project may have several benefits for the intended clients of this project, the Collins Lab scientists. By achieving high accuracy, low computational intensiveness, and user friendliness, this product can significantly aid in reducing the expenditure of time and effort by Collins Lab experimentalists by helping experimentalists prioritize combinations to test, thereby reducing the size of experimental screening assays. This product may also aid experimentalists in validating combinations that have been identified as possible candidates through experimental testing. Thus, the integration of this model into the workflow of experimentalists will accelerate the discovery of combination antibiotic therapies and will further the research efforts of the Collins Lab.

The final product's ability to achieve the accuracy, computational-intensiveness, and user-friendliness goals also has broader non-technical impacts. As discussed in Section 3.1, each of these three broad goals has impacts on non-technical contexts, including public and global health and safety (Appendix 2). Successful combination antibiotic therapies are crucial for effectively combatting the growing public and global health threats posed by antibiotic resistance. Thus, it is crucial to create tools that enable experimentalists to easily, efficiently, and accurately accelerate the testing, validation, and deployment of antibiotic therapies (Appendix 2). By achieving high accuracy, low computational intensiveness, and user friendliness, the final product developed in this project contributes to efforts to improve global and public health (Appendix 2). Furthermore, these achievements may also grant the final product the ability to reduce the number of combination therapies to be experimentally screened, tested, and validated, which in turn could have broader environmental and economic benefits by preventing resource waste (Appendix 2). Finally, through the process used to develop the machine learning and the dataset used to train and test it, this model avoids being biased by social and cultural factors (Appendix 2), another crucial factor in its ability to create a positive impact.

A variety of technical standards and best practices in the field guided the development of this project. Through the design of the machine learning models and the user interface, this product addresses key technical standards regarding model training, parameter tuning, evaluation, and usability [48] (Appendix 1). The achievement of most of the accuracy specifications by the model likewise allows the final product to address accuracy-related technical standards [48] (Appendix 1). In addition to these technical standards, the "Fairness, Accountability, and Transparency in Machine Learning" or FAT ML guidelines outline best practices in the field [44] (Appendix 1). The development of an accessible and easily usable interface in this project aids in addressing the best practices outlined by these FAT ML guidelines (Appendix 1).

Additional work may be performed in the future to further develop the final product, broaden its applications, and improve its performance. First, the generalizability of the model's predictions must be tested by assessing its performance when run on independent external datasets. In this project, the model was trained and tested with the Kulesa et al. dataset [30], so the logic it acquired to make predictions regarding combination synergy is based on patterns found in this dataset. In order to ensure that the model is able to extend what it has learned to other datasets, it is crucial to test its performance with external datasets. Second, it is also crucial to perform the user-friendliness assessment of the final product with a larger sample size. In the timeline of this project, three Collins Lab experimentalists were able to test the final product. However, testing the product with a greater variety of experimentalists, possibly also including those outside the Collins Lab, and making improvements to the user experience based on their feedback is essential in ensuring that this product is able to benefit experimentalists of diverse backgrounds and skillsets. A third way to broaden the application of the product created in this project is to extend the approach to other bacterial species. The model created in this project makes predictions regarding the synergistic activity of combination therapies against *E. coli*. Since different bacteria demonstrate widely varying responses to antibiotic therapies, independent models must be trained and tested for different bacterial species. Thus, creating similar models for other common resistant bacterial strains may be beneficial for broadening the applications of the product. Finally, performing interpretability analysis of the machine learning model developed in this project may be an extremely beneficial future step. Interpretability analysis aids in understanding the rationales behind a machine learning model's predictions. In this case, an interpretability analysis may aid in understanding which substructures in a combination most strongly influence its success or failure as a synergistic combination. Such findings may inform future efforts to design more effective combination therapies.

With future analyses and refinements such as these, the product created in this project has the potential to create numerous impacts on the broader community of scientists involved in antibiotics research as well as clinicians and medical personnel. By providing a way to predict the antibacterial synergy of a combination, this product can offer a way for scientists designing combination antibiotic therapies to rapidly screen and validate combinations. Interpretation of the model's rationales for predictions may grant scientists new insights for designing effective combination therapies. These impacts may aid in accelerating the development of pharmaceutical agents that clinicians can use to combat antibiotic resistance. The predictions made by the machine learning model may offer clinicians combination therapies that could be used to treat patients exhibiting antibiotic resistance more effectively. Combination therapies are also used for treating other diseases and infections beyond bacterial infections. If expanded for a broader scope, this model may support scientists in their efforts to develop combination therapies in a variety of medical fields.

8 Acknowledgements

I would like to thank my faculty advisor, Professor Jim Collins, for giving me the opportunity to work on this amazing project. I would also like to thank my advisor, Jackie Valeri, for her continuous support and invaluable help through every stage of this project. Thank you so much for helping me troubleshoot all my crazy bugs, talking through so many different ideas for every stage of this project, and always encouraging me. Thank you, Jackie, for being the best mentor I could ask for! I would additionally like to thank my thesis reader, Professor Sean Eddy, for offering me valuable feedback on my initial project proposal and my thesis presentation.

Next, I would also like Collins Lab scientists Dr. Erica Zheng, Dr. Aarti Krishnan, and Ian Andrews for testing the installation process and user interface in this project and for offering immensely helpful feedback for refining the overall user experience.

I would also like to thank the ES100 faculty and staff for their support throughout this project. I would especially like to thank my section leader, Dr. Melissa Hancock, for her support and guidance throughout the whole course. Thank you, Melissa, for your invaluable advice with improving the communication and presentation of my project, for pointing me towards the best resources, and for your weekly feedback that kept me on track through the year. I would also like to thank Dr. Linsey Moyer and Dr. Nishant Sule for their feedback on my Design Review and final thesis presentation. Thank you also to Dr. Nishant Sule for providing guidance on computational resources and helping me set up a Google Colab workspace to run my larger models.

Finally, I would like to thank my ES100 classmates, my friends, and my family for their support and encouragement and for listening to me talk about machine learning for months on end!

9 References

[1] A. E. Clatworthy, E. Pierson, and D. T. Hung, “Targeting virulence: a new paradigm for antimicrobial therapy,” *Nat. Chem. Biol.*, vol. 3, no. 9, pp. 541–548, Sep. 2007, doi: 10.1038/nchembio.2007.24.

[2] M. I. Hutchings, A. W. Truman, and B. Wilkinson, “Antibiotics: past, present and future,” *Curr. Opin. Microbiol.*, vol. 51, pp. 72–80, Oct. 2019, doi: 10.1016/j.mib.2019.10.008.

[3] “Antibiotic / antimicrobial resistance,” *Centers for Disease Control and Prevention*, Jul. 20, 2020. https://www.cdc.gov/drugresistance/ (accessed Aug. 09, 2020).

[4] “New report calls for urgent action to avert antimicrobial resistance crisis,” *World Health Organization*, Apr. 29, 2019. https://www.who.int/news/item/29-04-2019-new-report-calls-for-urgent-action-to-avert-antimicrobial-resistance-crisis (accessed Sep. 22, 2021).

[5] A. Baronia and A. Ahmed, “Current concepts in combination antibiotic therapy for critically ill patients,” *Indian J. Crit. Care Med.*, vol. 18, no. 5, pp. 310–314, May 2014, doi: 10.4103/0972-5229.132495.

[6] K. Bush and P. A. Bradford, “β-Lactams and β-Lactamase Inhibitors: An Overview,” *Cold Spring Harb. Perspect. Med.*, vol. 6, no. 8, p. a025247, Aug. 2016, doi: 10.1101/cshperspect.a025247.

[7] R. J. Ferreira, V. Aguilar, A. M. Villamil Giraldo, and P. M. Kasson, “Simulation-guided engineering of antibiotics for improved bacterial uptake,” Microbiology, preprint, Oct. 2020. doi: 10.1101/2020.10.08.330332.

[8] J. D. Romano and N. P. Tatonetti, “Informatics and Computational Methods in Natural Product Drug Discovery: A Review and Perspectives,” *Front. Genet.*, vol. 10, p. 368, Apr. 2019, doi: 10.3389/fgene.2019.00368.

[9] M. Der Torossian Torres and C. de la Fuente-Nunez, “Reprogramming biological peptides to combat infectious diseases,” *Chem. Commun.*, vol. 55, no. 100, pp. 15020–15032, 2019, doi: 10.1039/C9CC07898C.

[10] J. M. Stokes *et al.*, “A Deep Learning Approach to Antibiotic Discovery,” *Cell*, vol. 180, no. 4, pp. 688-702.e13, Feb. 2020, doi: 10.1016/j.cell.2020.01.021.

[11] C. W. Coley, R. Barzilay, W. H. Green, T. S. Jaakkola, and K. F. Jensen, “Convolutional Embedding of Attributed Molecular Graphs for Physical Property Prediction,” *J. Chem. Inf. Model.*, vol. 57, no. 8, pp. 1757–1772, Aug. 2017, doi: 10.1021/acs.jcim.6b00601.

[12] J. H. Yang *et al.*, “A White-Box Machine Learning Approach for Revealing Antibiotic Mechanisms of Action,” *Cell*, vol. 177, no. 6, pp. 1649-1661.e9, May 2019, doi: 10.1016/j.cell.2019.04.016.

[13] K. Yang *et al.*, “Analyzing Learned Molecular Representations for Property Prediction,” *J. Chem. Inf. Model.*, vol. 59, no. 8, pp. 3370–3388, Aug. 2019, doi: 10.1021/acs.jcim.9b00237.

[14] J. Vamathevan *et al.*, “Applications of machine learning in drug discovery and development,” *Nat. Rev. Drug Discov.*, vol. 18, no. 6, pp. 463–477, Jun. 2019, doi: 10.1038/s41573-019-0024-5.

[15] S. Jaeger, S. Fulle, and S. Turk, “Mol2vec: Unsupervised Machine Learning Approach with Chemical Intuition,” *J. Chem. Inf. Model.*, vol. 58, no. 1, pp. 27–35, Jan. 2018, doi: 10.1021/acs.jcim.7b00616.

[16] D. C. Elton, Z. Boukouvalas, M. D. Fuge, and P. W. Chung, “Deep learning for molecular design - a review of the state of the art,” *Mol. Syst. Des. Eng.*, vol. 4, no. 4, pp. 828–849, 2019, doi: 10.1039/C9ME00039A.

[17] S. Ekins *et al.*, “Exploiting machine learning for end-to-end drug discovery and development,” *Nat. Mater.*, vol. 18, no. 5, pp. 435–441, May 2019, doi: 10.1038/s41563-019-0338-z.
[18] G. Landrum, “Fingerprints in the RDKit,” p. 23.
[19] C. W. Coley *et al.*, “A graph-convolutional neural network model for the prediction of chemical reactivity,” *Chem. Sci.*, vol. 10, no. 2, pp. 370–377, 2019, doi: 10.1039/C8SC04228D.
[20] S. Chandrasekaran *et al.*, “Chemogenomics and orthology-based design of antibiotic combination therapies,” *Mol. Syst. Biol.*, vol. 12, no. 5, p. 872, May 2016, doi: 10.15252/msb.20156777.
[21] M. Cokol, C. Li, and S. Chandrasekaran, “Chemogenomic model identifies synergistic drug combinations robust to the pathogen microenvironment,” *PLOS Comput. Biol.*, vol. 14, no. 12, p. e1006677, Dec. 2018, doi: 10.1371/journal.pcbi.1006677.
[22] K. B. Wood, K. C. Wood, S. Nishida, and P. Cluzel, “Uncovering Scaling Laws to Infer Multidrug Response of Resistant Microbes and Cancer Cells,” *Cell Rep.*, vol. 6, no. 6, pp. 1073–1084, Mar. 2014, doi: 10.1016/j.celrep.2014.02.007.
[23] N. M. Smith *et al.*, “Using machine learning to optimize antibiotic combinations: dosing strategies for meropenem and polymyxin B against carbapenem-resistant Acinetobacter baumannii,” *Clin. Microbiol. Infect.*, vol. 26, no. 9, pp. 1207–1213, Sep. 2020, doi: 10.1016/j.cmi.2020.02.004.
[24] J. Li, X.-Y. Tong, L.-D. Zhu, and H.-Y. Zhang, “A Machine Learning Method for Drug Combination Prediction,” *Front. Genet.*, vol. 11, p. 1000, Aug. 2020, doi: 10.3389/fgene.2020.01000.
[25] S. K. Kim *et al.*, “Identification of drug combinations on the basis of machine learning to maximize anti-aging effects,” *PLOS ONE*, vol. 16, no. 1, p. e0246106, Jan. 2021, doi: 10.1371/journal.pone.0246106.
[26] H. Julkunen *et al.*, “Leveraging multi-way interactions for systematic prediction of pre-clinical drug combination effects,” *Nat. Commun.*, vol. 11, no. 1, p. 6136, Dec. 2020, doi: 10.1038/s41467-020-19950-z.
[27] Q. Liu and L. Xie, “TranSynergy: Mechanism-driven interpretable deep neural network for the synergistic prediction and pathway deconvolution of drug combinations,” *PLOS Comput. Biol.*, vol. 17, no. 2, p. e1008653, Feb. 2021, doi: 10.1371/journal.pcbi.1008653.
[28] W. Jin, R. Barzilay, and T. Jaakkola, “Discovering Synergistic Drug Combinations for COVID with Biological Bottleneck Models,” *ArXiv201104651 Cs Q-Bio*, Nov. 2020, Accessed: Aug. 09, 2021. [Online]. Available: http://arxiv.org/abs/2011.04651
[29] G. Landrum, *RDKit*. 2010.
[30] A. Kulesa, J. Kehe, J. E. Hurtado, P. Tawde, and P. C. Blainey, “Combinatorial drug discovery in nanoliter droplets,” *Proc. Natl. Acad. Sci.*, vol. 115, no. 26, pp. 6685–6690, Jun. 2018, doi: 10.1073/pnas.1802233115.
[31] Q. Liu, X. Yin, L. R. Languino, and D. C. Altieri, “Evaluation of Drug Combination Effect Using a Bliss Independence Dose–Response Surface Model,” *Stat. Biopharm. Res.*, vol. 10, no. 2, pp. 112–122, Apr. 2018, doi: 10.1080/19466315.2018.1437071.
[32] J. D. Durrant and R. E. Amaro, “Machine-Learning Techniques Applied to Antibacterial Drug Discovery,” *Chem. Biol. Drug Des.*, vol. 85, no. 1, pp. 14–21, Jan. 2015, doi: 10.1111/cbdd.12423.
[33] G. Arango-Argoty, E. Garner, A. Pruden, L. S. Heath, P. Vikesland, and L. Zhang, “DeepARG: a deep learning approach for predicting antibiotic resistance genes from

metagenomic data," *Microbiome*, vol. 6, no. 1, p. 23, Dec. 2018, doi: 10.1186/s40168-018-0401-z.

[34] "What to consider when choosing a laptop for machine learning (2020)," *AI Kenya*. https://kenya.ai/what-to-consider-when-choosing-a-laptop-for-machine-learning/ (accessed Sep. 26, 2021).

[35] A. Lee, "Why and How to Use Pandas with Large Data," *Towards Data Science*, Nov. 03, 2018. https://towardsdatascience.com/why-and-how-to-use-pandas-with-large-data-9594dda2ea4c (accessed Sep. 27, 2021).

[36] *Scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/

[37] G. Rodola, *Psutil*. [Online]. Available: https://pypi.org/project/psutil/

[38] "System Usability Scale (SUS)," *usability.gov*. https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html (accessed Sep. 29, 2021).

[39] M. Quirós, S. Gražulis, S. Girdzijauskaitė, A. Merkys, and A. Vaitkus, "Using SMILES strings for the description of chemical connectivity in the Crystallography Open Database," *J. Cheminformatics*, vol. 10, no. 1, p. 23, Dec. 2018, doi: 10.1186/s13321-018-0279-6.

[40] A. Capecchi, D. Probst, and J.-L. Reymond, "One molecular fingerprint to rule them all: drugs, biomolecules, and the metabolome," *J. Cheminformatics*, vol. 12, no. 1, p. 43, Dec. 2020, doi: 10.1186/s13321-020-00445-4.

[41] S. D. Axen, X.-P. Huang, E. L. Cáceres, L. Gendelev, B. L. Roth, and M. J. Keiser, "A Simple Representation of Three-Dimensional Molecular Structure," *J. Med. Chem.*, vol. 60, no. 17, pp. 7393–7409, Sep. 2017, doi: 10.1021/acs.jmedchem.7b00696.

[42] National Library of Medicine, "Ampicillin," *PubChem*. https://pubchem.ncbi.nlm.nih.gov/compound/Ampicillin#section=InChI-Key (accessed Mar. 26, 2022).

[43] "Feed-Forward networks," *Neural Networks*. https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Architecture/feedforward.html (accessed Sep. 28, 2021).

[44] "Principles for accountable algorithms and a social impact statement for algorithms," *FAT ML*, 2016. https://www.fatml.org/resources/principles-for-accountable- algorithms (accessed Jan. 20, 2022).

[45] *PyTorch*. [Online]. Available: https://pytorch.org/

[46] S. Sharma, S. Sharma, U. Scholar, and A. Athaiya, "□ACTIVATION FUNCTIONS IN NEURAL NETWORKS," vol. 4, no. 12, p. 7, 2020.

[47] W. Jin, "icml18-jtnn." https://github.com/wengong-jin/icml18-jtnn (accessed Mar. 17, 2022).

[48] "IEEE Standard for Technical Framework and Requirements of Trusted Execution Environment based Shared Machine Learning," IEEE. doi: 10.1109/IEEESTD.2021.9586768.

[49] "Technical AI standards." NIST, Aug. 06, 2021. Accessed: Jan. 20, 2022. [Online]. Available: https://www.nist.gov/artificial-intelligence/technical-ai-standards

[50] "IEEE SA - AIS Standards." IEEE Standards Association. Accessed: Jan. 20, 2022. [Online]. Available: https://standards.ieee.org/initiatives/artificial-intelligence-systems/standards.html#p7000

[51] O. Clermont, S. Bonacorsi, and E. Bingen, "Rapid and simple determination of the escherichia coli phylogenetic group," *Appl. Environ. Microbiol.*, vol. 66, no. 10, pp. 4555–4558, 2000.

[52] N. de C. Stoppe *et al.*, "Worldwide Phylogenetic Group Patterns of Escherichia coli from Commensal Human and Wastewater Treatment Plant Isolates," *Front. Microbiol.*, vol. 8, p. 2512, Dec. 2017, doi: 10.3389/fmicb.2017.02512.

[53] J. A. Ayukekbong, M. Ntemgwa, and A. N. Atabe, “The threat of antimicrobial resistance in developing countries: causes and control strategies,” *Antimicrob. Resist. Infect. Control*, vol. 6, no. 1, p. 47, Dec. 2017, doi: 10.1186/s13756-017-0208-x.

10 Appendix

Appendix 1. Relevant Engineering Standards

Since the field of artificial intelligence is a quickly evolving field, engineering standards in this field are largely still under development. For example, the National Institute of Standards and Technology (NIST) is engaging in an ongoing effort to collaborate with government and industry stakeholders to outline a set of technical artificial intelligence standards [49]. Similarly, the majority of the IEEE standards related to machine learning and artificial intelligence are still in developmental phases [50]. However, a few standards and guidelines for best practices in the field have been created and discussed by leading experts.

Although not fully relevant to the contents of this project, the *IEEE Standard for Technical Framework and Requirements of Trusted Execution Environment based Shared Machine Learning* is one of the few existing technical standards in the field of machine learning that contains guidelines relevant for this project [48]. One of the key guidelines in this standard regarding data management is the need for the model to be able to support and manage shared data, i.e. it must be able to acquire, collect, process, and explore data [48]. Keeping this in mind, this design product has ensured that it has the capability to acquire data from the file path specified by a user, collect all the data, process it in order to convert the data to the required input format for the model, and use the model to make explorative conclusions regarding the data. Many of the algorithm management requirements detailed in this standard [48] were also kept in mind for this design project. For example, this standard details that model training capability is essential, model parameter tuning (i.e. hyperparameter optimization in this project) should be supported, and that recommended model evaluation metrics include auROC, precision, and recall [48]. Additional guidelines that are relevant for the development of this project include that the system should permit the model to make predictions after being trained and that it should have various usability features (i.e. the user interface best suited for Collins Lab experimentalists in this project) [48].

Although not an official standard, another set of recommendations known as "Fairness, Accountability, and Transparency in Machine Learning" or FAT ML, is a key set of guiding principles in the field of artificial intelligence [44]. A few of the FAT ML principles in particular were particularly influential in this project. First of all, FAT ML emphasizes the importance of accuracy [44], which motivated the emphasis on accuracy-related metrics when outlining the technical specifications for this project. The FAT ML principle of auditability describes the importance of providing third party individuals the opportunity to examine the algorithm as well as interact with it through an appropriate interface [44]. This aspect of the FAT ML principles was taken into account in this project by creating a user-friendly user interface in the platform and format that is most appropriate for the technical backgrounds of the intended clients of this project. FAT ML also has a principle of fairness, indicating that the decisions of an algorithm should not discriminate across different demographic groups [44]. In this project, since the model relies on data obtained from bacterial assays, without the involvement of human test subjects, the underlying data from which the model learned to make its predictions is as objective as possible. Thus, this should hopefully prevent biased predictions with regards to human-specific factors such as demographic factors. The specific strain of *E. coli* used in the experiments used to generate the ground truth data used by this model was the K-12 strain of *E. coli* [30], a primarily lab-based strain that falls into the *E. coli* phylogenetic group A [51]. The impact of socioeconomic factors on the phylogenetic distribution of *E. coli* strains has been previously observed [52]. However, since the *E. coli* phylogenetic group A, to which the K-12 strain belongs, is most prevalent group worldwide [52], the K-12 is the best possible choice for use in a ground truth dataset with regards to minimizing bias in the dataset and therefore the model.

Appendix 2. Non-Technical Contexts

Public health, safety, and welfare

Rapid discovery of new and effective combination antibiotic therapies that can successfully synergize and produce antibacterial responses without being impeded by antibiotic resistance is crucial for public health, safety, and welfare. Effective antibiotic therapies are crucial for medical personnel to effectively treat bacterial infections, and with the growing public health threat posed by antibiotic resistance, it is essential to develop methods of aiding experimentalists in more rapidly discovering effective antibiotic therapies. With this context in mind, it was essential for the model developed in this project to 1) provide accurate predictions that experimentalists can reasonably rely on to accelerate the testing, validation, and deployment of antibiotic therapies and 2) be delivered to experimentalists in a platform that they can easily and efficiently use to perform the tasks and obtain the information that is most useful to them. Thus, this context played a strong role in the development of the technical specifications for this project and specifically the focus on ensuring high accuracy, low computational intensiveness, and user-friendliness (see Section 3).

Global factors

In addition to domestic public health and safety, the antibiotic resistance crisis and the challenges it poses have the same health and safety impacts on a global scale as well. In fact, antimicrobial resistance crisis is even more severe in developing countries due to lack of public awareness, adequate technologies, and adequate regulatory mechanisms [53]. Thus, this global context played a role very similar to the public health context in the motivation and development of this project and its technical specifications (see Section 3).

Cultural factors

As mentioned in Appendix 1, a key part of the FAT ML guidelines is ensuring that algorithms are not biased across different demographic groups, including different cultural groups. Thus, it was important in this project to ensure that cultural and other demographic factors do not influence the predictions made by the model. The choice of a ground truth dataset (see Section 4.2) that relies solely on bacterial assays rather than any data relating to specific patients helps meet this goal by ensuring that the only data available for the model to learn from is free of any human-specific factors. This does come at a cost, however, as *in vitro* assay performance does not always translate directly to *in vivo* bactericidal activity. Hence, subsequent experimental validation of the outputs of this model is still crucial.

Social factors

Like with the cultural context, it is also important that ML algorithms avoid being biased by other social factors as well. Thus, once again, the use of a ground truth dataset (see Section 4.2) relying on bacterial assays rather than patient data helps meet this goal by ensuring that the model is not influenced by social factors. As mentioned in Appendix 1, the specific strain of *E. coli* used in the experiments used to generate the ground truth data used by this model was the K-12 lab-based strain [30] that falls into the *E. coli* phylogenetic group A [51]. As previously described, socioeconomic factors can influence the phylogenetic distribution of *E. coli* strains, but since the *E. coli* phylogenetic group A is most prevalent [52], the K-12 is the best choice for use in a ground truth dataset (see Section 4.2) with regards to minimizing bias in the dataset and therefore the model.

Environmental factors

Currently, during experimental testing of combination therapies, a vast majority of the tested combinations are not successful synergistic combinations. These large-scale assays result in enormous expenditure of resources and materials. Especially considering that most of these assays do not have successful outcomes, there is a fair amount of resource wastage that occurs during experimental testing. Using a computational model to reduce the number of combinations to be tested and especially increase the portion of tested combinations that are successful could alleviate this resource wastage. This context motivated the need for high accuracy in the model and specifically a low false positive rate in order to reduce this resource and labor wastage. Thus, this context motivated many of the broad goals for this project as well as the technical specifications (see Section 3).

Economic factors

The same resource wastage described above that has environmental impacts also poses economic costs. As the efficacy of existing antibiotics decreases further in the next few years and decades, large-scale testing of single-agent and combination therapies will only become more and more necessary. As this need for testing escalates, resource consumption and therefore the costs of performing this testing will also escalate. Therefore, like with the environmental impacts, it is important to alleviate these economic costs through the use of accurate models. Thus once again, this context motivated the need for high accuracy in the model and specifically a low false positive rate in order to reduce this resource and labor wastage (see Section 3).

Appendix 3. Data and Code Availability

https://github.com/mythriambatipudi/CombosPrediction

The above repository is currently not publicly viewable, but access can be made available upon request.

Appendix 4. User-Friendliness Evaluation Survey

https://forms.gle/HW8gYRrhvkeRh14e9

Appendix 5. Installation Guide

1. If you do not already have conda, install Anaconda and add it to your path:
    a. wget http://repo.continuum.io/archive/Anaconda3-2020.02-MacOSX-x86_64.sh
    b. sudo bash Anaconda3-2020.02-MacOSX-x86_64.sh
    c. export PATH=~/anaconda3/bin:$PATH
    d. vim ~/.bashrc and add "export PATH=~/anaconda3/bin:$PATH" to the last line
    e. source ~/.bashrc or reboot
    f. Check that the install worked by trying: conda –version. There should be an output with your conda version.

2. Install Python 3.7 if you do not already have it: conda install -c anaconda python=3.7

3. Find the "anaconda3" folder on your computer and navigate into the "envs" folder.

4. Download the environment.yml and the requirements.txt files and drag them into this "envs" folder.

5. Create and activate a virtual environment called abx-env with python 3.7 from the environment.yml file.

    a. Navigate to the anaconda3/envs folder in your terminal if you are not already there (for example, cd opt/anaconda3/envs/)
    b. conda env create -f environment.yml
    c. conda activate abx-env

6. Finish with a few last independent package installations:
    a. conda install -c rmg descriptastorus
        i. When it asks you if you would like to proceed, type in y for yes
    b. pip install -r requirements.txt
    c. pip install torch
    d. pip install torch-geometric

   e. pip install torch-sparse
   f. pip install torch_scatter

7. Download the folder “Final_FFN_Package” and drag it into the directory in which you are working, for example Documents/combos.

8. From your terminal, navigate out of the anaconda3/envs directory and navigate to the “Final_FFN_Package” folder. For example, if the “Final_FFN_Package” is located in a folder called “combos” within your Documents folder:

   a. To navigate out of the anaconda3/envs directory: cd ../..
   b. To navigate to “Final_FFN_Package”: cd Documents/combos/Final_FFN_Package

9. Making sure the abx-env conda environment is still activated (you should see (abx-env) in front of every line in your terminal), you should be able to use the command jupyter notebook, which should launch the Jupyter window in your web browser. Open up the notebook combos_prediction_UI.ipynb. The notebook is located inside the Final_FFN_Package folder.
   a. When the notebook is opened, change the kernel by going to Kernel → Change Kernel → Python [conda env: abx-env]

10. To close the notebook, press Ctrl+C in terminal. All changes made to files in your current directory are saved to your local machine.

11. Every time you wish to return to this notebook, simply activate the abx-env conda environment as described in Step 5b and then launch jupyter notebook.

Troubleshooting:

Problem: Any kind of errors that indicate that a molecule may be invalid (e.g. if you get NoneType or index out of range errors when generating RDKit features or when generating molecular fingerprints)
Solution: Make sure that all the SMILES strings in your dataset are valid SMILES strings. Especially make sure that there are no extra spaces or punctuation marks before and after the SMILES strings.

Problem: The progress bar is progressing very slowly, and it is taking very long for the cells to finish running.
Solution: If your dataset contains too many combinations, it will take a very long time to generate features from all the SMILES strings and to convert all the SMILES strings to molecular fingerprints. If you want to obtain results faster, consider trimming down the dataset you are inputting to a smaller subset of SMILES strings.

Appendix 6: Budget

BOM (Bill of Materials)	Unit Cost $	Unit	# of Units	Total $	Exact or estimated?
Google Colab Pro+	$53.11	1 month	5	$265.55	Exact

Please do your best to fill in the following details on your budget	Total $	Exact or estimated?
Total Development cost: everything that was spent on your project including prototypes, transportation to research locations, renting of equipment, orders from a lab you worked at, your own money etc...?etc...	$265.55	Exact
What is the minimum cost to make one prototype of your project ($0 is an option)?	$0	Exact
Total cost of items purchased through the *Active Learning Labs (ALL), if any*	$265.55	Exact
Total cost covered by the *Harvard Research* Lab(s) you are affiliated with, if any	$0	Exact
Total cost of items purchase personally, if any	$0	Exact
Total cost covered by a *non-Harvard lab* and/or company, if any	$0	Exact

If you had to use CNS space and equipment please answer the following quesitons:
Why did your project require the use of CNS facilites? **NA**
Roughly much time did you spend using CNS facilities? **NA** [specify hours or days]
What equipment did you need to use? **NA**
Who did you interact with at CNS and how? **NA**
If you are affiliated with a research lab that arranged for your use of CNS facilities, please specify which lab (and resarch administrator who arranged your access, if you know) **NA**
If you didn't work through a research lab directly, did the Active Learning Lab sponsor/arrange your use of CNS resources? **NA**
What could we have done better to make your CNS experience more productive? **NA**

Please list below all material used in **ALL** that were ***not*** accounted for in your budget / made available to you at no cost, for e.g.: