Drug Discovery With Graph Neural Networks
— Final Report —

Yi siang Ong, Elizabeth Bates, Hongye Liu, Abir Sridi, Qi Li, Yikang Li
{yi.ong21, elizabeth.bates21, hongye.liu20, a.sridi, qi.li21, yikang.li20 }@imperial.ac.uk

Supervisor: Dr Paul Bilokon
Course: COMP70048, Imperial College London

March 12th, 2023

1 Introduction

Drug discovery pipelines are complex and costly in all aspects. Whether in the early stages of identifying biological targets or the later phases of safety and efficacy analyses, each drug in development must undergo lengthy research and testing. A key part in assessing a drug's viability is in the ADMET (Absorption, Distribution, Metabolism, Elimination, Toxicity) properties studies [1]. The topics of molecule solubility and toxicity are highly relevant in the drug discovery field and critical in a molecule's bioavailability. As such, they are the topic of our software project. It has been suggested that poor pharmacokinetic features and toxicity are the main issues responsible for the 40% attrition rate of candidate drugs in development [2]. Sensical molecules that are water-insoluble or exhibit toxic traits are not suitable drug candidates; an early understanding of these critical properties allows for less time-wasting and reduced spending in the pharmaceutical research industry.

Advances in AI, particularly in Deep Learning and Machine Learning (ML), offer the potential to quicken various tasks in drug discovery, given excellent data and appropriate application. There is existing software available such as SwissADME [3] which can perform tasks like solubility prediction. However, their methodology uses Support Vector Machines (SVMs) whilst our project explores the use of Graph Neural Networks (GNNs), more specifically, Graph Convolutional Networks (GCNs). The use of GCNs is still relatively new in the drug discovery context, though results in a classification task show GCN models producing models comparable to or outperforming that of the descriptor-based SVM models [4].

1.1 Project Specification

On reflection of the existing software in the fields of molecule solubility and toxicity and the complexity of the drug development process, the team decided that our project ought to focus on being able to predict these two features using a novel GCN architecture and deploying it as an accessible web app. We decided that two forms of solubility should be displayed, the continuous logS value and a probability that this molecule is water-soluble enough for further drug progression. Both these outputs could be useful to someone looking to decipher if a molecule is worth further exploring. The demographic this web app is tailored to covers anyone in the academic field of drug discovery or pharmaceutical chemistry. Our main goals were to create a web app that was intuitive, quick and produced accurate indications of a molecule's predicted solubility and toxicity.

2 Methodology

2.1 Literature Review

2.1.1 SMILES

The "Simplified Molecular-Input Line-Entry System" (SMILES) [5], [6] is a symbolic language for describing the structure of chemical molecules (the atoms and bonds of molecules) in the form of short text strings of ASCII characters and are unique molecular representations. SMILES strings can be imported into most molecular structure editors to be converted back to a 2D representation or a 3D model, making building computer databases relatively simple. There are other online text languages to represent molecules, including InChl, introduced by IUPAC. SMILES is, however, more widespread because it is easier to read by those in the drug development field and especially because it is in use across lots of existing molecular chemistry software. For instance, SMILES allows one to translate the structure given in figure 1, Nicotine ($C_{10}H_{14}N_2$) [7], into a linear representation of the molecule so that a computer program can understand the structure. Here is the SMILES Notation for Nicotine:

$C_{10}H_{14}N_2$: CN1CCC[C@H]1c2cccnc2

2.1.2 LogS

LogS is defined as a standard solubility unit corresponding to the 10-based logarithm of the solubility of a molecule measured in mol/litre [8]. There is no clear defined boundary in terms of logS values,

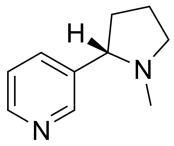

Figure 1: Nicotine molecule structure [7]

which means a molecule is appropriately soluble to be a drug molecule. There is a slight variation across existing work on this. However, most seem to use the metric that compounds with a logS of 0 or higher are highly soluble, those in the range of –2 to 0 are soluble, those in the range of –4 to –2 are slightly soluble and insoluble if less than –4 [9] [3]. Hence, a molecule is soluble (1) if $\log S \geq -4$ and insoluble (0) otherwise.

2.1.3 The Nuclear Receptors-Androgen Receptor (NR-AR)

The Tox21 Platform Toxicology provides a library of chemical data, screened in high-throughput assays against a panel of 12 different target-based biological pathways. These involve two broad groups of adverse outcome pathways: the Nuclear Receptors (NR) and the Stress Response (SR) pathways. NR assays are classified into seven subtasks pathways, including the Androgen Receptor (AR) that we have retained as a characteristic of toxicity: its value is 1 if the molecule is toxic and 0 otherwise [10].

2.1.4 Mathematics of Graph Neural Networks

In the context of data structures, a graph can be expressed as $G = (V, E, A)$, containing a set of nodes $i \in V$ and a set of edges $e_{ij} \in E$. Nodes V represent vertices whilst edges E represent the connections between two nodes (vertices). For example, if nodes a and b are connected, then $e_{ab} = 1$. Otherwise, $e_{ab} = 0$. The corresponding Adjacency Matrix, A, contains the connection information [11].

Node features h_i, h_j in a graph can represent atom types; edge features e_{ij} indicate the type of bonds between graph features. The graph features g specify the molecule energy.

To understand the main GCN architecture, one must first be familiar with general Graph Neural Networks. A general Graph Neural Network can be expressed as $H^{l+1} = f(A, H^l)$, where A represents the adjacency matrix and H^l represents the node features.

In this project, the main idea of Graph Neural Network is to comprehensively search and output the function f to precisely project the node feature and adjacency matrix to the next node feature. The GCN architecture is more complex than the general Graph Neural Network and can be written as: $H^{l+1} = \sigma(D^{-1/2} * \tilde{A} * (\tilde{D})^{-1/2} * \theta)$, where D represents the degree matrix, $\tilde{A} = A + I$ represents the adjacency matrix and its self-loop, $\tilde{D} = D + I$ represents the degree matrix and its self-loop, σ is the nonlinear activation function, and θ is the trainable parameter.

The architecture of GCNs ensures higher precision and accuracy by adding the self-loops or self-connections to the adjacency matrices. In this case, all nodes are connected to themselves, and thus the sources node's embedding during message aggregation is considered.

2.1.5 Using Graph Neural Networks

GNNs are useful architectures in that they can handle different datasets that are too complex or abstract that can not be solved by the classic Convolutional Neural Network. For example, CNNs can only handle the Euclidean Structure Data like natural images. However, they would be useless if we used non-Euclidean Structure data like the social network as the input data. However, a GNN is capable of handling both datasets by manipulating the dataset into graph form [12].

As previously mentioned, a graph is composed of nodes and edges. Both types of data can be transformed into graphs. For example, a social network could be transformed into graphs with the

user node and relationship edges. A natural image can also be converted into graphs with the pixel node and adjacency edges. In our case, the drug bank dataset is suitable for transferring to a graph dataset.

Another useful trait of Graph Neural Networks is the message-passing property. The semantic message-passing through different layers in a CNN is done by converging and combining the local images. The message-passing process is quite different for GNNs. Firstly, a node starts to send messages to other nodes. The given node also receives messages from other nodes and updates itself consequently. Each node will learn about its environment iteratively. In this case, any node message will contain many different messages from other nodes. Since there is no order within the nodes, the model is usually permutation-invariant. Therefore, message-passing can be achieved via many different paths and gathers most nodes' messages.

2.2 Development Methodology

To deliver our software, we adopted the Scrum methodology, which is known to be one of the best agile methods in use today. We opted for this choice as we were convinced that an agile Scrum process would better help control the project's schedule and status, increase the quality and efficiency of the output and cope better with project changes.

All team members have been developers, testers, and designers simultaneously. In addition to his role as Team Member, Yi Siang Ong was the Scrum Master, and our supervisor Dr Paul Bilokon was the Product Owner. We set a one-week sprint duration as the project was quite adaptive. We organised a consistent 2-hour group meeting on Mondays at 4 pm, where the scrum master checked everyone's progress, discussed the sprint outcomes and planned for the week ahead. The first hour was spent discussing what was well executed, and what ought to be improved in the next sprint, then the second was focused on planning the next sprint, discussing and allocating tasks amongst team members from the product backlog.

The scrum master ensured that the project included the whole team and often people worked collaboratively on tasks both as a group and within sub-groups. We also met our supervisor weekly on Microsoft Teams (Wednesdays at 1 pm). During the sessions, we presented the work we had done during the week, checked whether implementations had the features and quality and then made the final call to release this work, discussed issues faced and got helpful feedback. All feedback was fed back into our product development process. In addition to these weekly meetings, we organised daily brief online meetings. These daily meetings were for everyone on the team to be updated and be compatible with the sprint goal. It was also the time to voice any concerns regarding the sprint goal or any blockers. Our group meetings were conducted on Microsoft Teams, and we had regular chats on our Whatsapp group.

Trello was used as a workflow management tool to define, manage and improve our work. We saved the backlogs to Trello and broke the work down into small sections. Trello visualises completed tasks, what needs to be done and what is in progress, as well as the priority of each task in the form of "cards", making it easy to track short-term goals. Responsible team members can be added to cards, and cards can be moved to different lists, representing different stages of the process. We followed the Scrum framework and tackled complex adaptive issues with this tool while productively and creatively delivering high-level work.

2.3 Design and User Story

2.3.1 Design

We used various software tools to develop our web app for drug discovery. We used python 3 to write the backend, including data processing and AI model building. We decided to use python in this project because it is simple, consistent and suitable for collaborative implementation, especially since all our group members have practical experience with python. While doing data processing, we used libraries like pandas, Seaborn, NumPy, Matplotlib and scikit-learn, which are useful for data manipulation and analysis. In the model building, training and evaluating sections, we used libraries like PyTorch and PyTorch Geometric. PyTorch is an open-source machine learning framework, and

PyTorch Geometric is one of the most popular GNN libraries. We used PyTorch Geometric to pass geometric data into our GNN and design a custom MessagePassing layer.

Other essential libraries we used in the project include Wandb and DeepChem. Wandb is an experiment tracking tool for machine learning. When the training code is instrumented with Wandb, the background process collects valuable data about the model, such as hyperparameters and model performance metrics and outputs visual charts for analysing the model performance.

In terms of the web-framework library, we used Streamlit. This is a python based open-source application framework for machine learning and data science. With Streamlit, we integrated a frontend web page with various functions using the backend python files.

2.3.2 User Story

In this subsection, two user stories have been proposed, outlining how a potential user from our target demographics could interact successfully with our web app.

1. A research team from a pharmaceutical firm has collated lots of data to investigate in hopes of finding a new possible drug molecule for a given illness. They would like an approximate and quick understanding of the basic properties of the potential drug molecules such that they can rule out unsuitable ones. The research team can use our web app and upload a CSV file containing a list of SMILES molecules; an example input file can be downloaded such that the format can be matched by the user. The website has been designed such that the team can choose between predicting the LogS, solubility and toxicity of each molecule. Having both a LogS value and a general solubility probability gives the research team more of an indication of specific solubility such that they can make more informed decisions. The web service is free and open, making the early-stage drug discovery process simple and economical.

2. Graduate students would like to use our website to predict the properties of some drug molecules for their project. They can either input a single SMILES to make a quick prediction or upload a file. They can predict whether specific to toxicity or solubility and then download these outcomes for further analysis.

2.4 Data Handling

We worked with two datasets during this project. The first dataset was used to accurately predict the water solubility of drug molecules and consists of an XML dataset taken from the drugbank website [13]. The dataset consists of a total of 13,339 molecules. We started by parsing the XML file and picking the logS and SMILES elements of interest. There were often two different logS values available to pick, either experimental or predicted. We used exclusively experimental values of logS since these are drug properties that have been experimentally proven [14] and as the predicted values were described as sometimes inaccurate by DrugBank [15]. We proceeded to clean the data by replacing dummy data. In particular, we replaced infinite values with NaNs and then dropped columns having null values. We then transformed the logS feature into a float. Moreover, we added a solubility categorical feature: 1 if the molecule is soluble and 0 otherwise.

We ended up with 10,013 molecules containing 6223 labelled as soluble and 3790 labelled as non-soluble. The task is to determine logS for molecules and classify them into these two categories given their SMILES. It was clearly evident that our target class had an imbalance, which would mean our model would be substantially more familiar with one class than the other. This would result in a model with poor predictive performance, specifically for the minority class. The training data required modification to avoid simply predicting the majority class in all cases. To this end, we split our data randomly into 0.9 for training and 0.1 for testing and upsampled the training data such that the minority class matches with the majority class; in particular, we carried out a sample with replacement. This resulted in obtaining 5597 samples for each of the classes. We further divided the training dataset into training and validation with a ratio of $7:2$.

The second dataset is used to accurately predict the toxicity of drug molecules and consists of a CSV file collected from the website https://deepchemdata.s3-us-west-1.amazonaws.com/datasets/

tox21.csv.gz This dataset is composed of 7831 molecules and 14 features. 12 of the 14 features figured in this dataset could describe toxicity; we decided to retain 'NR-AR' as the toxicity feature because it comes with more samples than the other features. In the same way, as for solubility, we cleaned the dataset by replacing infinite values with NaNs, dropping lines having at least one null value. We then added a toxicity categorical feature: 1 if the molecule is toxic and 0 otherwise, which gave us 290 molecules labelled as toxic, and 6374 labelled as non-toxic. The task is to classify them into these two categories given their SMILES. Since Our target class had an imbalance, we randomly split the data into two separate files corresponding to a 'train' (0.9) and a 'test' (0.1) set. We then upsampled the training data so that the minority class matches with the majority class; we carried out a sample with replacement. This resulted in 5735 samples for each of the classes. We further divided the training dataset into training and validation with a ratio of 7 : 2.

2.5 Machine learning pipeline

Since our project was meant to implement software that could accomplish the classification and regression tasks, the main architecture that we used was a simple GNN. The performance of our software had shown that a GNN was capable of handling the drug molecule datasets and finishing these tasks. Moreover, the quantitative results given by the GNN also showed that a GNN could perform well with the Drug Bank dataset.

2.5.1 Data preprocessing: DeepChem library

The DeepChem toolchain [16] was used throughout the project. This library is aimed mainly at deep-learning projects covering topics like drug discovery, quantum chemistry and biology, and hence was ideal to use in our software. DeepChem contains many valuable tutorials and datasets; the Tox21 dataset was used in the training of the toxicity model to predict if a molecule is likely to be toxic or not. It provided much insight into how we ought to approach a project combining chemistry and GCNs. The tutorials demonstrated the use of SMILES and appropriate methods of molecule featurisation and a valuable introduction to graph convolutions. After the data was processed, the DeepChem toolkit was vital to convert the SMILES data into 2D molecule representations; since we used *torch_geometric* models, we had to create a custom dataset. The *custom_dataset_classification_toxic.py* read the CSV file and enabled *edge_features*. This resulted in a NumPy array containing a featurised representation of SMILES of a drug molecule. It further transferred the array into a graph that was suitable to be used by *torch_geometric*.

2.5.2 Model Architecture

For a simple GCN model, we first obtained the node embeddings, then fed them forward into the model function with dropout layers, aggregated message passing and finally applied the final dense layer to output the results. The following figure 2 displays our simple GCN model architecture.

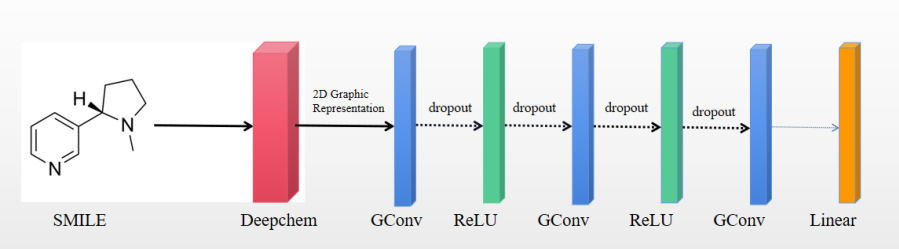

Figure 2: Simple GCN Model Architecture.

2.5.3 Best Model Selection and Model Evaluation

This subsection explains our model selection strategy by hyperparameter tuning and our model evaluation metrics. In our project, there were three main tasks we aimed to utilise the model described

above, and each model was optimised for its respective task:

1. Predicting logS value from a SMILES string using the regression model.
2. Predicting solubility of a SMILES string using the classification model.
3. Predicting toxicity of a SMILES string using the classification model.

The evaluation metric used to optimise the tasks was different. For the regression task, we used Explained Variance Score (EVS) to measure its performance and tried to optimise it. We used EVS as the evaluation metric because EVS could measure the discrepancy between a model's prediction and actual data. It would clearly state the part of the model's total variance that is explained by factors that are actually present rather than error variance [17]. We used the Area Under Curve (AUC) as the evaluation metric for the classification task. The AUC can be interpreted as the definite integral of a curve that describes the variation of a drug concentration in blood plasma as a function of time in the field of pharmacokinetics [18]. AUC is a better metric than accuracy to evaluate classification model as it does not bias on size of test data.

Next, the methodology we used to select the best set of hyperparameters for $N = 20$ epochs was aided with the Wandb tool. Wandb tries different training parameter values in order to find a good set of weight values. In order to perform hyperparameter tuning and select the best model, we used the Bayes hyperparameter tuner provided in the Wandb. Bayesian optimisation is a sequential design strategy for global optimisation of both regression and classification functions. It is usually employed to optimise expensive-to-evaluate functions. Not only does it save the time of manually tuning the hyperparameter, but it also increases the precision of the model steadily. We first created a Wandb project, and then 50 Bayes search were run. Within the search, the model weights and biases were reinitialised, trained and optimised via Adam optimiser for 20 epochs, and each set of hyperparameters was logged together with its evaluation metric such that a parallel coordinate visualisation of every run could be displayed as shown in figure 3. Finally, we used the best set of hyperparameters to retrain the model and stored the model weights and biases using the early stop strategy in a pickle file.

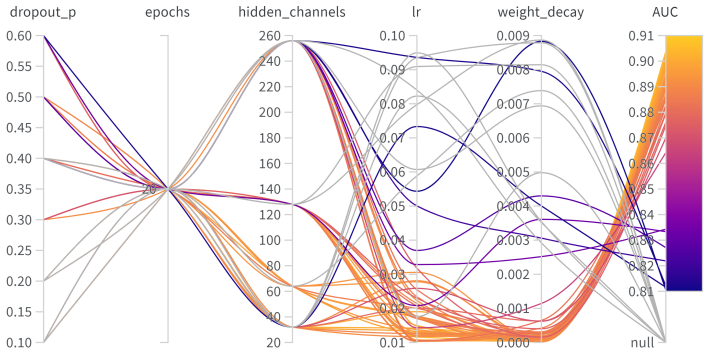

Figure 3: Example of hyperparameter sweeps (Solubility Classification model) using Wandb tool

1. Predicting logS of a SMILES string (regression task):

The best set of hyperparameters for predicting logS value we have obtained consists of hidden channels of 128, dropout probability of 0.2, a learning rate of 0.0023 and weight of decay $-3.2e^{-5}$. After obtaining the optimised set of hyperparameters, the model was retrained and evaluated for the best possible performance using EVS. The higher the EVS, the better performance by our model as the model's discrepancy would be lower. The following displays the results of the performance for training and validation: From figure 4 and figure 7, as the number of epochs increases, both training and validation EVS also increase, whereas the losses decrease. As mentioned before, we applied the early stop method to select the best epoch number at 60 epochs to avoid the model from overfitting. As a result, the values of the training and validation EVS that we have obtained are 0.759 and 0.757, respectively, whereas the values of the training and validation loss that we have got are 0.71 and 0.715,

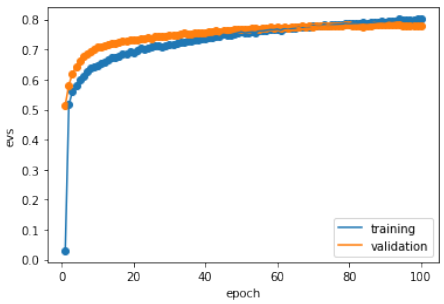

Figure 4: EVS v.s. epochs for predicting logS value

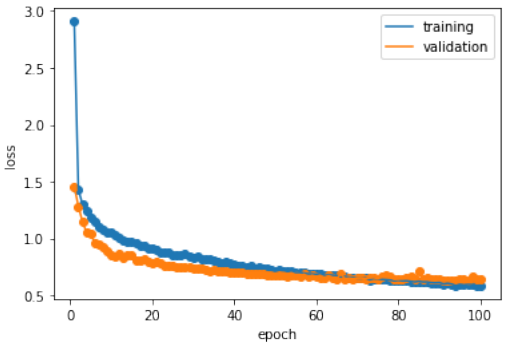

Figure 5: Loss v.s. epochs for predicting logS value

respectively. After testing the best model with the test dataset, we obtained the test EVS being 0.752, which is similar to the training and validation ones. In addition, the test Root Mean Squared Value (RMSE) was also calculated at 0.767. This demonstrates that our model performs well because there was no considerable reduction between test results and training results and a low RMSE value. It might be because our model was fair and had a restricted bias.

2. Predicting solubility of a SMILES string (classification task):

The best set of hyperparameters for this classification task we have obtained consists of hidden channels of 128, a dropout probability of 0.6, a learning rate of 0.0116 and a weight of decay -0.00007191. Similarly, the model was retrained and evaluated for the best possible performance with a different metric, the AUC score. The higher the AUC score, the better performance by our model as the higher capability of the classifier to distinguish between the positive and negative classes. The following displays the results of the performance of this classifier for training and validation: From

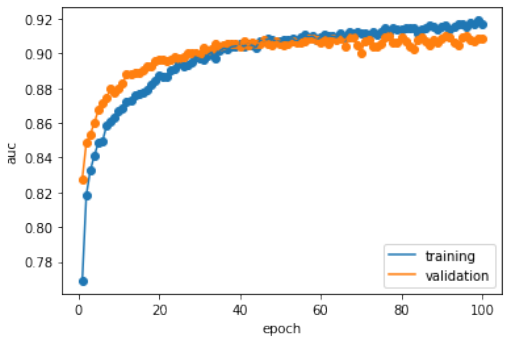

Figure 6: AUC score v.s. epochs for is soluble classification task

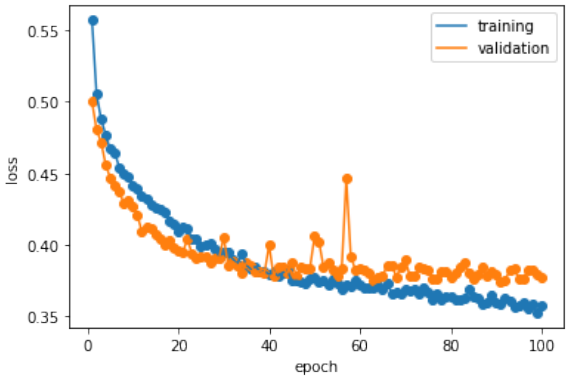

Figure 7: Loss v.s. epochs for is soluble classification task

figure 10 and figure 7, a similar trend is observed as the number of epochs increases; both training and validation EVS also increase, whereas the losses decrease. We applied the early stop method to select the best epoch number at 60 epochs to avoid the model from overfitting. Consequently, the values of the training and validation AUC score that we have obtained are 0.910 and 0.908, respectively. On the other hand, the training and the validation loss that we have obtained are 0.369 and 0.383, respectively. After testing the best model with the test dataset, we obtained the test AUC score being 0.914, which is similar to the training and validation ones. This demonstrates that our model performs well because there was no considerable reduction between test results and training results. It may be because our model was fair, which had a restricted bias. In addition, we calculated the confusion matrix, accuracy, precision, and recall, to assess the final performance of the model. The confusion matrix is shown in figure 8. The accuracy value is 0.83, which means that the predictions of the solubility of drug molecules are pretty reliable. The value of precision and recall are 0.92 and

0.78, respectively. The high precision means that our model is conservative, giving relevant results - positive solubility. Also, the recall value implies that our model often predicts the positive solubility class.

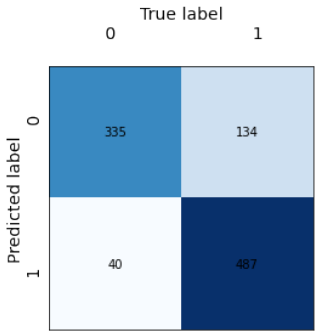

Figure 8: Confusion Matrix for Solubility Classification

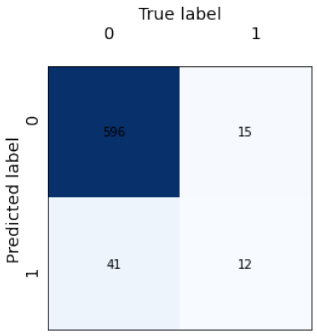

Figure 9: Confusion Matrix for Toxicity Classification

3. Predicting toxicity of a SMILES string (classification task):

The best set of hyperparameters for predicting toxicity we have obtained consists of hidden channels of 256, a dropout probability of 0.3, a learning rate of 0.01607 and a weight of decay $-4.371e^{-5}$. After obtaining the optimised set of hyperparameters, the model was retrained and evaluated for the best possible performance. We used AUC to measure the model's performance to distinguish between classes. The higher the AUC, the higher the ability of the classifier to distinguish between the positive and negative classes. The following displays the results of the performance for training and validation:

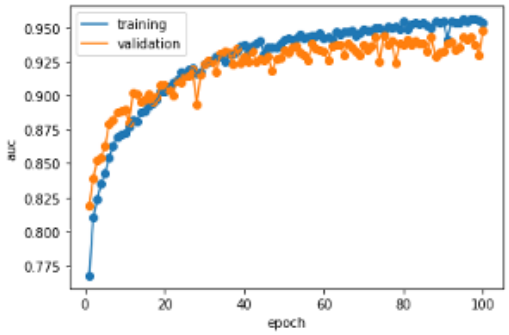

Figure 10: AUC scores v.s. epochs

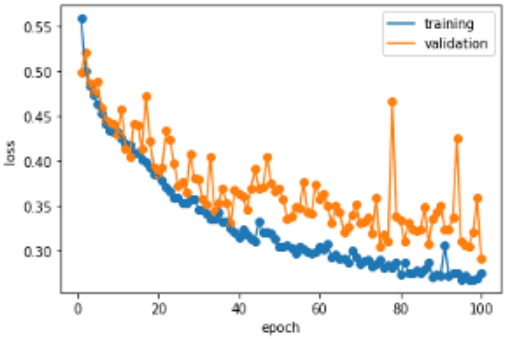

Figure 11: Loss v.s. epochs

From figure 10 and figure 11, as the number of epochs increases, both training and validation AUC also increase. As mentioned before, we applied the early stop method to select the best epoch number at 40 epochs to avoid the model from overfitting. As a result, the training and validation AUC values that we have obtained are 0.933 and 0.925, respectively, whereas the training and validation loss are 0.324 and 0.361, respectively. After testing the best model with the test dataset, we obtained the test AUC being 0.74, which has a reduction from the training and validation ones. In addition, we calculated the confusion matrix, accuracy, precision, and recall, to assess the final performance of the model. The confusion matrix can be seen in figure 9. The accuracy value is 0.92, which means that the predictions of the toxicity of drug molecules are pretty reliable. The value of precision and recall are 0.23 and 0.44, respectively. The low precision means our model is not very conservative in the positive toxicity class. Also, the recall value implies that our model does not often predict the positive toxicity class.

We have achieved better results in terms of the AUC score obtained on the testing dataset, precision and recall when dealing with solubility by comparison to toxicity. This is due to an accentuated imbalance in the toxicity data before upsampling (see section 2.4).

3 Application and architecture

3.1 Frontend Development

There were a few options of which web app framework to use, such as Flask or Dash, but Streamlit was identified as the most appropriate and visually appealing. It is beneficial since we started with a python code base, and building a web UI via Streamlit is more efficient and quicker than the other frameworks. The web app needed to be easy to use and intuitive, so the initial goal whilst developing the frontend was to make a basic interface where a file could be uploaded, the user press 'predict' and subsequently, a table would be displayed with corresponding predictions. This was followed by fringe testing to ensure that all the edge cases were caught and a suitable message was displayed to the user if they did not interact appropriately with the web app. Once the web app functioned adequately fundamentally, the focus changed to adding additional features to enhance the app. Examples of this were adding a way of inputting data, not just by file but by being able to type in a SMILES string too for a quick individual prediction. We presented our website to Dr Namshik Han, who leads the computational biology group at the Milner Therapeutics Institute University of Cambridge. He suggested that users need an intuitive understanding of the drug molecules' structures. We further developed our website to handle more complex features based on his feedback. The results were further illustrated with molecules' names, graphical structures and predictions. If the molecules can be featurised by the DeepChem library, their graphical structures are shown for users to understand the drug molecules intuitively; otherwise, an 'unknown molecule' is shown. The 2D graphical representations would be helpful since the SMILES strings are hard to read, but the images would make molecule differentiation easier for the user. It is of note that there's a limit of 50 molecules in the input file; the model can handle more than this, however the loading time was long and so file size was capped for a smoother user experience. A button was also added for users to download the prediction outcome for further use. Various help messages and a clear website description were finally added, followed by more fringe testing before it was deployed to Heroku.

3.2 Backend Handling Methodology

Backend typically refers to the part of the application to handle and store users' data. Since our website design was purely focused on predicting the outcome from the user input data, there was no need to store their data, e.g. personal information or input data, but to handle the preprocessing part of the input data. Our methodology to handle and preprocess the input data was discussed as follows.

Firstly, we received the data returned from the frontend and processed it into the data structure required by the predicted model. Since a small number of molecules cannot be processed by DeepChem and RDKit, we searched its index and defined the results as Nan. Each time the user makes a prediction, the backend would temporarily create a folder to store the temporary data processed for the predicted model. Then input the data into our trained model for prediction. Finally, the results are returned to the front end, and the molecules are visualised by the mols2grid package. After the prediction, the temporary data folder is automatically deleted to release the cache. Following this, we used Streamlit to quickly build the frontend and backend to achieve a demo demonstration. Finally, the project was deployed to Heroku to achieve public network access.

3.3 Version Control System and Continuous Integration

The Version Control System (VCS) we are utilising in this project is *git*, via the git repository management system *GitLab*. Having a VCS is crucial for projects that involve collaborative coding. From a git repository, a developer can pull the most up to date version before contributing additional work to the project. Git also enables us to track all the changes that have been made so far, so it allows us to go back to the previous working version of the codebase if part of the code was accidentally crashed. In addition, we have further developed our continuous integration approach to ensure that any new commits pushed would not break the existing functionality. All the codes pushed to GitLab are tested within the CI pipeline to ensure the product is kept fully functional before deployment. This step is crucial as various group members worked on different parts of the codebase. Furthermore,

this method assisted us in reducing numerous times of conflicts within the codebase. One of the advantages of using GitLab to manage our git repository is that we could easily set up a CI pipeline that links to our configured runner in the virtual machine. We then could write a script to run a series of commands which consists of building a valid environment in our virtual machine and automatically performing unit tests to ensure that our main functionality works as intended. Besides building and testing, the CI helped us deploy our application to Heroku to make our website a public domain.

3.4 Quality Assurance

Traditional testing systems such as verification and validation approaches were used in our Quality Assurance (QA) process. A robust QA process is essential to boost the users' confidence in our product. The QA plan comprises two components:

1. Verification

The verification process is to ensure our final product meets the specifications set out in the Team Agreements Report, where the specifications are translated into quantifiable and testable requirements. However, due to the nature of this project involving black-box models, it is not easy to provide an exact specification of what the product should do. On top of that, there is a time constraint for this project; therefore, after discussion with PGTA during the consultation meeting, it was sufficient to only perform some necessary verification tests on the product's main functionality, which was to predict the given SMILES properties.

Since our product consists of only one main module (*model_predict.py*), the testing approach would mainly focus on unit testing that module. Furthermore, our product was intended to be user friendly such that everything from input data to displaying results could be done on one page. Therefore, no integration testing was required for the product. We were also able to use acceptance testing as we asked in our weekly meetings with our supervisor if he was satisfied with the current functionality of our product.

Unit tests are fast and easy to implement. This means that the unit tests enable us to identify and locate bugs quickly and easily such that we can apply appropriate fixes. It is also a form of regression testing since we test that if the new version breaks the old functionality, it would be reflected on the Gitlab CI pipeline at each push to the repository. As mentioned, the goal of the unit tests is to ensure our main features work correctly and to compare our approximations with the ground truth. Following are the unit tests, and these tests take less than a few seconds to run:

- GCN output Tests: We often had a clear idea of what a specific output should be – for example, a Numpy array output by the model prediction. We have prepared several CSV files for testing three of the main functions; given a valid SMILES input, if the user chooses to predict logS, they should output an array with list elements of class "float", whereas class "int" for IsSoluble and IsToxic. Also, the array should output a NoneType if the input SMILES cannot be featurised by the DeepChem library, we had contacted DeepChem author regarding the issue of valid SMILES that cannot be featurised and still awaiting his response.

- Ground Truth Tests: We have also implemented a ground-truth test to the classification task to predict IsSoluble and IsToxic as they provide much higher confidence in predicting the truth as shown during the model testing.

2. Validation

Verification testing only revealed errors within the frame of tests of the main functionality. Any other issues may not be clearly visible. To make our product more robust, we took a user-centric view based on the user story and validated our product from the user's point of view to maximise its utility to the end-user. To capture all other aspects not tested in the CI pipeline, we performed validation tests within the group and consider the following aspects of our product essential to the end-user:

- Prediction time: The layers within the model are tested such that the prediction time is shortened at the same time, avoiding the trade-off of the model's accuracy. Also, a small icon on the top right corner of the website indicates the run time as the prediction is in the process after the user selects the prediction target and inputs their data.

- Sources of possible error: Incorrect user inputs or usage are probably the main sources of error missed by the CI pipeline. Hence, we have thrown numerous situations at the product to ensure the exception handling of our product. For instance, if the user inputs an invalid SMILES, the graph could not be displayed, and an error message would be shown.

3.5 Deployment Methodology

In terms of deployment, we utilised tools like Docker and Heroku.

Docker is an open-source tool for creating, managing, deploying, and running applications in containers. Unlike Heroku, which has to run in its cloud environment, Docker can be installed on a laptop, server, or cloud-based environment like Amazon Web Services (AWS). Docker mainly consists of a Dockerfile, a set of commands used to build a Docker image. Dockerfiles also represent the final configuration of a Docker image. A Docker image is a multi-layered file containing all the resources, dependencies, data, files, and settings needed to run an application. Every time a Docker image is started, it creates a container of the same application, providing easy scalability.

The other tool, Heroku, is a multilingual platform where developers can deploy applications in almost any programming language by using various buildpacks. Heroku allows instant scalability; it could be easily scaled up or down by increasing or decreasing the number of Dynos. Heroku is straightforward to set up, implement and deploy. Heroku manages hardware, software, and other resources, allowing developers to focus more on developing their applications.

Our specific pipeline was realized by connecting Docker and Heroku through the CI/CD of GitLab. We built the CI/CD process by configuring the *.gitlab-ci.yml* file. These included establishing the virtual machine environment, testing the project, and the final deployment. Through *heroku.yml*, the final project was deployed to Heroku in a containerised manner according to the Dockerfile to improve the portability and security of the application so that users could access the final application using the public network. Containers are virtual environments that run on a single operating system kernel, enabling applications to run using a fraction of the resources required by a virtual machine or bare metal computer system. Containers allowed us to scale applications quickly and easily transit from development to production. When the entire pipeline is built, CI/CD will automatically update and deploy the app whenever the project team members submit the latest code and merge it into the main branch.

3.6 Website Link and Demonstration

To access the repository, please follow the url: `https://gitlab.doc.ic.ac.uk/g21mscprj13/msc-ai-group-project.git`. To access our web app, please follow the url: `https://icl-drug-discovery.herokuapp.com/` The homepage offers a short description of the web app, its possible uses, how to use it and how to interpret the results given, see figure 12. The user has the option to either upload a file containing SMILES data, see figure 13 or type in a single SMILES string, see figure 14, in the sidebar on the left of the page. Following this, one can decide which of the options in the drop-down menu they would like to predict, see figure 15. The results of the prediction are displayed as a table with an accompanying molecule graphics table displaying the 2D molecular structure of the molecules being assessed, see figure 17. If the user does not interact appropriately with the web app, appropriate messages will be displayed to let them know how to proceed; see messages in the figure 16.

4 Group Work

We collaborated both as a group and within sub-groups. These cooperative methods benefited improving performance and progression, increasing productivity, creating a flexible approach to our work, and making overall problem-solving easier. During our regular weekly meetings, we reviewed our progress and decided on the direction our work should take next; we made sure everyone was clear on everyone's tasks, dealt with issues that arose and set new objectives based on our progress and feedback from our supervisor. The distribution of tasks within the sub-groups was linked to the particular interests of each member of the team. We ensured that each group member had responsibility and accountability

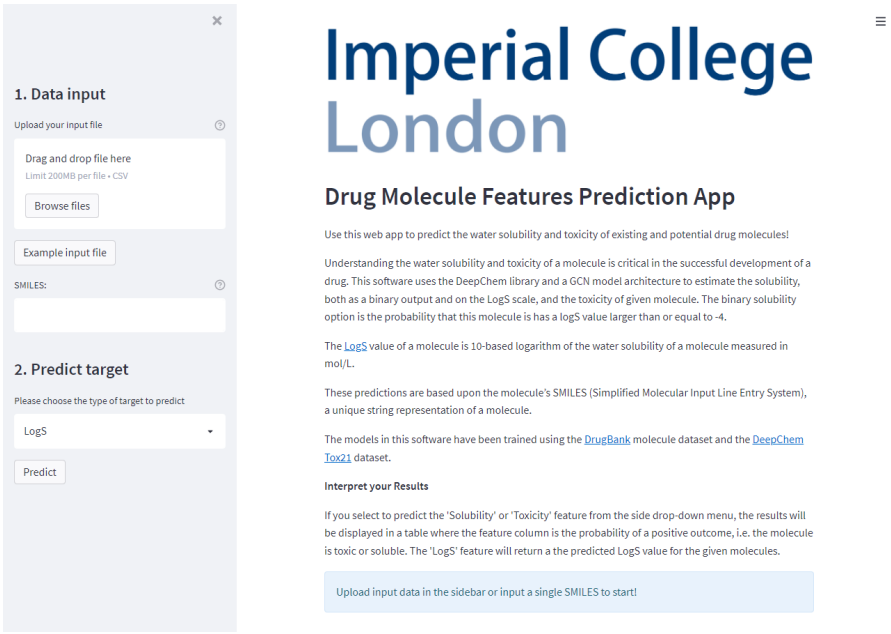

Figure 12: The Web App Homepage

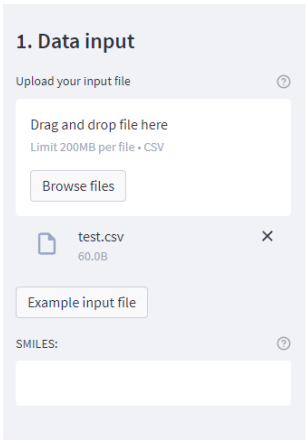

Figure 13: Option to upload a file

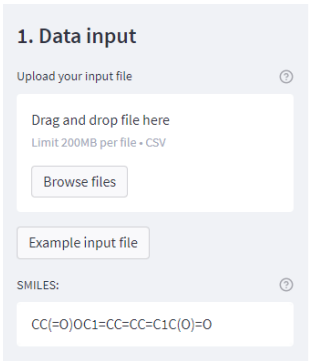

Figure 14: Option to input SMILES string directly

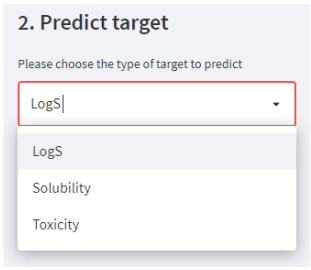

Figure 15: Drop-down menu options

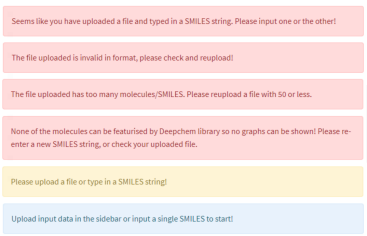

Figure 16: Possible messages in response to user activity

for their task within an agreed time frame. We also encouraged each other to voice any concerns or progress blockers and share any resources, information or knowledge.

We started the project with a preliminary research stage which was not very structured in terms of teamwork. The following different stages involved tasks that we decided to split between pairs or small groups. Table 1 displays the contribution of each team member throughout the project.

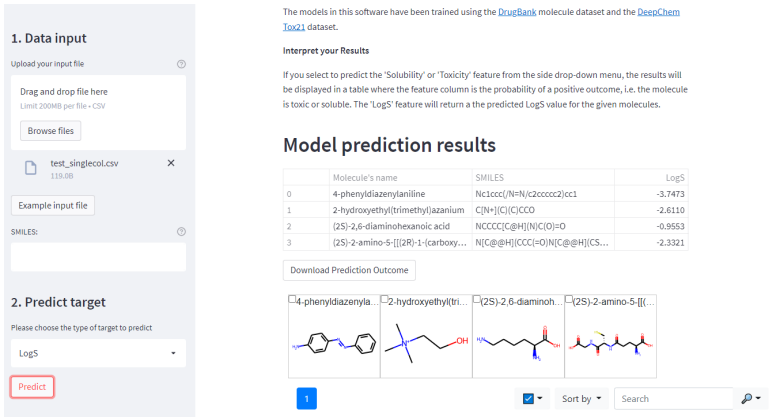

Figure 17: The Web App Homepage

Project Sections	Main Tasks	Primarily Responsible
Research	Understanding existing work and SOTA Research as well as what a useful contribution in this field would look like.	All
Data and Preprocessing	Data collection, interpretation, cleaning, formatting, translation into our working environment.	Abir, Hongye, Yi, Qi
Models	Graph Neural Network model building (classification and regression models) and hyperparameter tuning to find best models.	Yi, Hongye, Liz, Yikang, Abir
Frontend Development	Deciding on app framework, designing website look, integrating models into Streamlit, assessing usability	Liz, Abir, Qi, Yikang
testing	Writing PyTests for various aspects of the source code.	Yi, Abir
Deployment	Deploying to Heroku and GitLab integration.	Yi, Hongye

Table 1: Contribution of each team member throughout the project.

5 Final Product Summary

In summary, the project's general goal was to develop an app that allows users to predict the water solubility and toxicity of drug molecules. We designed the backend with a model of GCN to predict the two features of the drug molecules and streamlit is used to link the frontend and backend. Finally, the project is deployed to Heroku so that users can access it using the public network. Our app meets the needs of potential users. First, it achieves the essential functions required – predicting solubility and toxicity of drug molecules. Second, it is free, accessible and easy to use; it could reduce costs for the user working in the drug discovery field. Third, our app can predict the features within seconds. This could reduce the time spent by drug developers in being able to eliminate unsuitable molecules. Last, our user interface was seen by Dr Namshik Han and further modified under his suggestions – we added graphical structures with molecules' names to give the user a better intuitive understanding.

5.1 Description of the final product

The final product of our project is a web app, which is mainly implemented by the Streamlit package. The web app is mainly used to predict the water solubility and toxicity of existing and potential drug molecules. There are types of input data that can be accepted. The first input data type is a CSV file which contains several different SMILES. The second type of data is actual individual SMILES strings. After we plug the data into the app web, we can apply the GCN to predict the logS, solubility, and toxicity of the SMILES.

5.2 Challenges

There were several challenges we faced during the project. The first challenge was the data processing problem. As computing students, we have little experience in preprocessing the molecule dataset, which requires a strong background in chemistry. In this case, we had to understand the structure of the molecule dataset. Moreover, we also had to learn how to use the DeepChem library since GCNs can only accept a graph from input data. The DeepChem library is mainly used to transform the drug molecule datasets into some graphs datasets. The second challenge we faced was the optimisation problem. Since we were implementing software to solve different tasks on different datasets, we spent a long time on hyperparameter tuning. However, the training time was quite long for each type of parameter. In this case, we used the Bayesian optimisation method, which could improve the performance of models iteratively. For accomplishing the bayesian optimisation method, we used the Wandb package, which can automatically train the models with different hyperparameters. We have saved a lot of time by using the Wandb package. The final challenge we met was implementing the web app. Almost all of us are just beginners in software implementation since we are all AI students. It meant that time had to be allocated to become familiar with various packages and deployment techniques before implementing the software. Although the streamlit package could handle both the frontend and backend software parts, we had to spend time learning how to connect Heroku with the implemented web app.

5.3 Product Evaluation

We have evaluated our product by ourselves, alongside Dr Bilokon and Dr Han too. We believe that our product has satisfied all our expectations initially and even performs better than we had anticipated.

There are three main parts that we made progress on. First of all, we have expanded our output in more detail. Initially, only quantitative results were shown after predicting, but this was adjusted to display the exact molecule name and type of the SMILES. Secondly, our software initially only accepted an input file; this is not particularly efficient for a user only needing to test one molecule. The web app was changed so the user can submit a CSV file or manually type the SMILES string. Thirdly, we changed the output from binary classes to actual probability. For example, initially, our software will show 0 or 1 to state whether a molecule is toxic or not. Now our software shows the probability of the SMILES being toxic. This increases the range of results and indicates the confidence our model has of each result.

6 Conclusion

We built a web app from scratch that meets the set goals in a flexible manner. Developed with rigour, our product tackles a real-life problem. It accurately predicts water solubility, logS and toxicity of drug molecules which are essential features for successful drug development. Users may interact with the software through a user-friendly interface. We are confident that our product is an appropriate solution to the problem we sought to solve, is in a production-ready state and has potential to be very useful to our target demographic.

Lastly, there is some future work that could follow on from the final state of this project:

- Alternative graph-based models could be implemented, such as Graph Attention Network, which could be evaluated to seek which model would be better at performing which task.
- Further work could be done on the model's architecture to enhance its performance and its time to predict the results. In addition, an optimisation algorithm on our app could be designed to reduce the time required to display our model's result and load the webpage.

References

[1] Longfei Guan et al. "ADMET-score–a comprehensive scoring function for evaluation of chemical drug-likeness". In: *Medchemcomm* 10.1 (2019), pp. 148–157.

[2] A Ganesan and K Barakat. "Solubility: a speed–breaker on the drug discovery highway". In: *MOJ Bioequiv Availab* 3.3 (2017), pp. 56–58.

[3] Antoine Daina, Olivier Michielin, and Vincent Zoete. "SwissADME: a free web tool to evaluate pharmacokinetics, drug-likeness and medicinal chemistry friendliness of small molecules". In: *Scientific reports* 7.1 (2017), pp. 1–13.

[4] Han Altae-Tran et al. "Low data drug discovery with one-shot learning". In: *ACS central science* 3.4 (2017), pp. 283–293.

[5] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *J. Chem. Inf. Comput. Sci.* 28.1 (1988), pp. 31–36.

[6] William J. Wiswesser. *A Line-Formula Chemical Notation*. Cromwell, 1954.

[7] *Simplified molecular-input line-entry system*. URL: https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system. (accessed: 07.04.2022).

[8] *Log S*. URL: https://dev.drugbank.com/guides/terms/log-s. (accessed: 24.04.2022).

[9] Murat Cihan Sorkun, Abhishek Khetan, and Süleyman Er. "AqSolDB, a curated reference set of aqueous solubility and 2D descriptors for a diverse set of compounds". In: *Scientific data* 6.1 (2019), pp. 1–8.

[10] Priyanka Banerjee et al. "ProTox-II: a webserver for the prediction of toxicity of chemicals". In: *Nucleic acids research* 46.W1 (2018), W257–W263.

[11] Benjamin Sanchez-Lengeling et al. "A Gentle Introduction to Graph Neural Networks". In: *Distill* (2021). https://distill.pub/2021/gnn-intro. DOI: 10.23915/distill.00033.

[12] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386.

[13] *Free drug data you can rely on*. URL: https://www.drugbank.com/academic_research?_ga=2.43101602.665110970.1650618464-1918904498.1641507603. (accessed: 22.04.2022).

[14] DrugBank. *Experimental property*. URL: https://dev.drugbank.com/guides/terms/experimental-property. (accessed: 07.04.2022).

[15] DrugBank. *Predicted property*. URL: https://dev.drugbank.com/guides/terms/predicted-property. (accessed: 07.04.2022).

[16] Bharath Ramsundar et al. *Deep Learning for the Life Sciences*. https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837. O'Reilly Media, 2019.

[17] James A Rosenthal. *Statistics and Data Interpretation for Social Work*. Springer, 2011.

[18] Hans H Maurer. "Multi-analyte procedures for screening for and quantification of drugs in blood, plasma, or serum by liquid chromatography-single stage or tandem mass spectrometry (LC-MS or LC-MS/MS) relevant to clinical and forensic toxicology". In: *Clinical biochemistry* 38.4 (2005), pp. 310–318.