Natzaret Gálvez Rísquez

GraphDTA for predicting drug–target binding affinity

Final Master's Thesis

directed by Dr. Francesc Serratosa

Master's Degree in Computer Security Engineering and Artificial Intelligence

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2024

Acknowledgements

This thesis would not have been possible without the generous help and guidance of my thesis supervisor, Dr. Francesc Serratosa, I would like to extend my sincere gratitude for giving me the opportunity to do this work and learn from it.

And I would like to express my thankfulness to my parents and relatives who have supported me during the thesis.

Abstract

The development of new drugs is costly, time consuming and often accompanied with safety issues. Drug repurposing can avoid the expensive and lengthy process of drug development by finding new uses for already approved drugs. To repurpose drugs effectively, it is useful to know which proteins are targeted by which drugs. Computational models that estimate the interaction strength of new drug–target pairs have the potential to expedite drug repurposing. Several models have been proposed for this task, between them GraphDTA. GraphDTA represents drugs as graphs and uses graph neural networks to predict drug–target affinity. To evaluate the effectiveness of GraphDTA's affinity prediction, we compared its performance with other established prediction algorithms, such as DeepDTA and WideDTA, using standard datasets (Kiba and Davis). The comparative analysis revealed that GraphDTA not only outperforms these models in predictive accuracy but also demonstrates superior adaptability when applied to diverse and uncontrolled datasets, like those from the URV. Furthermore, although GraphDTA did not achieve high performance without relying on SMILES representations, using 3D structural data instead of SMILES could potentially retain more vital molecular information that might be lost with SMILES. This approach of saving more important information of the molecules offers an avenue for future enhancement in our study and in the model. This possibility highlights the model's robustness and versatility, suggesting that with further development, it could handle different data types more effectively for drug-target affinity predictions.

Key words: GraphDTA, drug-target, binding affinity, prediction, SMILES, dataset.

Table of contents

Table of tables

Table of figures

1 Introduction

1.1 Context

The development of new drugs is an expensive and lengthy process, with costs averaging around billions of dollars and a timeline that can extend up to almost twenty years for FDA approval [1, 2]. Drug repurposing, which involves finding new therapeutic uses for already approved drugs, can circumvent much of this time and expense [2]. However, conducting these experiments in a laboratory setting is very costly and requires a significant amount of time to examine the affinity of a drug toward its targets because there are a vast number of compounds and targets. Thus is why, using computational methods ("in-silico" techniques) as a predicting tool for drug-target interaction is useful to identify potential drugs for further biochemical verification [3] due to speed up the process and greatly reduce the costs to find the drugs candidates for the target and vice versa.

However, effective drug repurposing requires understanding the interactions between drugs and their protein targets because drugs will interact with their target but also can interact with other targets, which often will be unknown and unintended [4]. Therefore, research on drug target affinity (DTA) prediction from computational approaches has been in development and has been studied for approximately twenty years [5, 6], leaving us with different computational methods of drug-target affinity calculation to be used by professionals and also some cases are available for the public, where they can be found uploaded and with a guide in GitHub.

One of this prediction methods is GraphDTA, which is a novel computational model designed to predict drug-target binding affinity, which is a measure of the strength of the interaction between a drug and its target protein. Unlike other models that represent drugs as 1D strings, like DeepDTA and WideDTA, GraphDTA leverages the natural graph structure of molecules. In this model, drugs are represented as molecular graphs, where atoms are nodes and bonds are edges. This graph-based representation allows the model to capture the intricate structural information of molecules, potentially leading to more accurate predictions of drug-target affinities [7]. Also, due its complexity in form, most models use 1D structure for protein representation, where it is the case for this method.

Due to the need to find the affinity of the drug(s) by component and target protein in a graph representation manner, GraphDTA offers a more accurate representation used to predict drug-target affinity.

This project aims to study the prediction results of GraphDTA in detail and compare them with those of other models. Subsequently, an algorithm will be developed to prepare the URV datasets to meet the input requirements of the GraphDTA model for the in-silico prediction of drug-target affinity. This approach will allow the model to be applied more broadly, beyond the Kiba and Davis datasets, thereby making the model adaptable to other types of datasets. Additionally, we will modify the GraphDTA algorithm to generate graphs directly from .sdf files for ligands, rather than converting these files to SMILES, in order to avoid information loss associated with the SMILES format. Finally, we will conduct a comprehensive discussion of all analyses and findings.

1.2 Objectives

The main objectives of this thesis are:

1. Conduct a comprehensive literature review to study the current methods for predicting drug-target interactions
2. Perform an extensive bibliographic search of various computational models for drug-target affinity prediction
3. Compare the performance of GraphDTA with other existing methods for drug-target affinity prediction
4. Apply GraphDTA to datasets generated at the URV, consisting of various pairs of the main protease of SARS-CoV-2 and a drug, to evaluate its effectiveness
5. Change GraphDTA in order to accept .sdf files and create graphs directly from those files instead of the creation and use of SMILES
6. Provide theoretical explanations of: Graph Convolutional Networks, Binding affinity prediction and graph regression

1.3 Memory structure

Having established the context and objectives, the work will proceed in alignment with these goals. First, we will examine the state of the art in drug-target interaction prediction methods and related computational models. This will be supported by an explanation of various important concepts associated with the thesis topic. Next, we will outline the workflows of the prediction models GraphDTA, DeepDTA and WideDTA for drug-target affinity. Afterward, we will apply GraphDTA to two datasets from URV to assess its performance on non-standard datasets, which will involve creating an algorithm to adapt these datasets for use with GraphDTA. Following this, we will modify GraphDTA to function without the use of SMILES strings, instead utilizing .sdf files directly for ligand representation. Finally, we will compare the performance of GraphDTA with other models (DeepDTA and WideDTA) to evaluate the effectiveness of different algorithms in predicting DTI, using the Kiba and Davis datasets. We will also analyze the results obtained from the non-standard datasets and evaluate the performance of the modified non-SMILES algorithm on the Fragalysis and PDB datasets (from URV).

1.4 Technology used

1.4.1 Visual Studio Code

Visual Studio Code (VS Code) is a powerful and lightweight code editor that is available for Windows, macOS, and Linux. It provides a variety of features designed to enhance the development experience across different programming languages, like JavaScript, Python, C++ and more [9]. In this project, we will use Visual Studio Code alongside Python Notes to streamline our development workflow when using GraphDTA algorithms. Python Notes is an extension that facilitates working with Python scripts and Jupyter Notebooks within VS Code. This combination provides an efficient and user-friendly environment for coding, testing, and running Python programs, making it easier to manage and analyze data.

1.5 Phases of the project

The master's thesis is divided into several phases, each focused on different aspects of developing and analyzing drug-target interaction (DTI) prediction models:

- **GraphDTA Model Analysis:** Conduct an in-depth analysis of the GraphDTA algorithm to evaluate its performance in predicting drug-target interactions for the Davis and KIBA datasets. This includes calculating the Concordance Index (CI) and Mean Squared Error (MSE), as well as assessing predictions for various graph neural network variants (GCN, GAT_GCN, GAT, GIN).
- **DeepDTA and WideDTA Models Evaluation:** Analyze the prediction results for DTI on the Davis and KIBA datasets using the DeepDTA and WideDTA models. This phase also involves creating an algorithm to generate the motifs file required for the Davis dataset when using the WideDTA model.
- **URV Dataset Preparation:** Develop a script to preprocess the URV datasets, converting them into a format compatible with GraphDTA. This ensures that the datasets meet all input requirements for GraphDTA's prediction model.
- **Direct Ligand Graph Construction in GraphDTA (no SMILES):** Modify the GraphDTA algorithm to allow direct reading of ligands from .sdf files and generating graph representations without relying on SMILES strings.
- **Final Report Preparation:** Compile a thorough report detailing the methodologies, results, and insights from the thesis, integrating all findings into a final document.

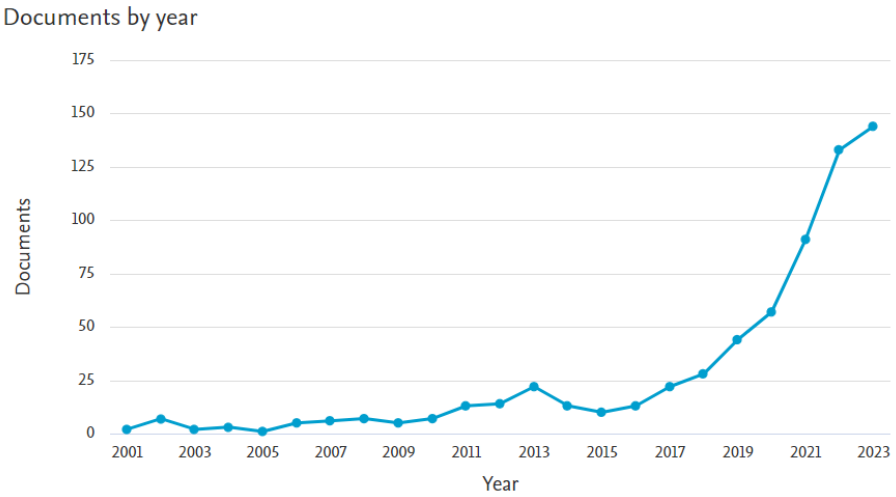
2 Prediction of drug-target affinity (DTA)

Drug-Target Affinity (DTA) prediction is a crucial aspect of modern drug discovery and development. Accurate prediction of the binding strength, or affinity, between a drug molecule and its target protein can significantly streamline the identification of potential therapeutic agents, reducing both the time and cost associated with experimental screening. This binding affinity refers to the strength of the interaction between a drug and its target protein, typically quantified by parameters such as dissociation constant (K_d), inhibitory concentration (IC_50), or binding energy [10]. High binding affinity usually indicates a strong interaction, which is desirable in drug development as it suggests a higher likelihood of efficacy at lower doses.

Accurate prediction of these binding affinities helps in prioritizing compounds for further development, thus playing a pivotal role in drug discovery pipelines. Traditionally, this domain has relied on empirical methods and high-throughput screening, but advances in computational methods have revolutionized the field, offering new avenues for precise and efficient predictions.

The growing body of research reflects the importance of DTA prediction in this area. Since the beginning of the 21st century, numerous articles have been published on the topic, with a significant contribution from Chinese authors, followed by India. Notably, the volume of publications related to DTA has been steadily increasing, particularly since the start of 2021, as evidenced by the analysis of documents available in the Scopus database [11]. The majority of these documents fall within the subject areas of 'biochemistry, genetics and molecular biology' and 'computer science', accounting for a total of 39.3% of the total publications. This is followed by 'chemistry', 'pharmacology, toxicology and pharmaceutics', and 'medicine', which

collectively make up 32.7% of the total documents. Where it is found that these are the main areas of interest for DTA prediction research to be developed in.



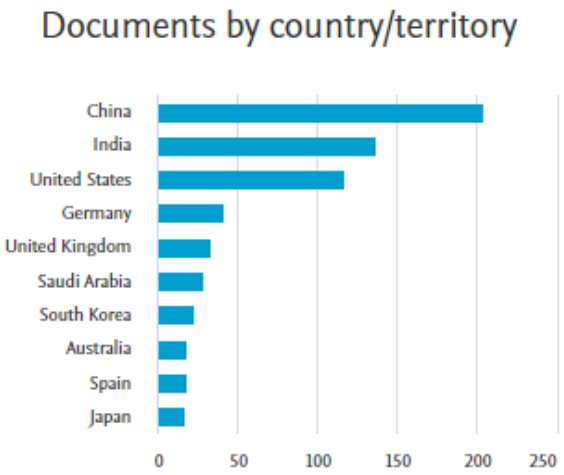
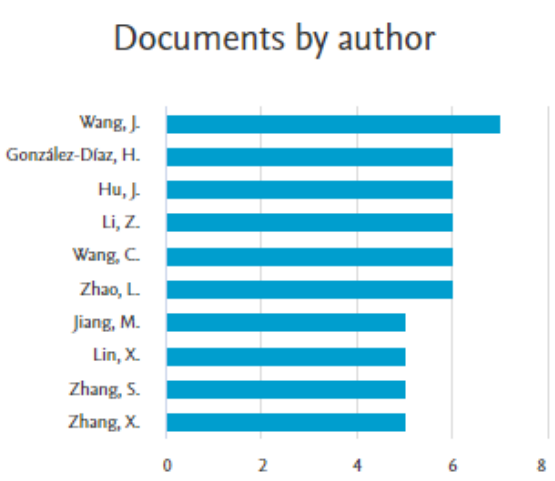


Figure 1. Documents by year and by country/territory for results of "Prediction of drug-target affinity" in Scopus [11] from 2001 to 2023.



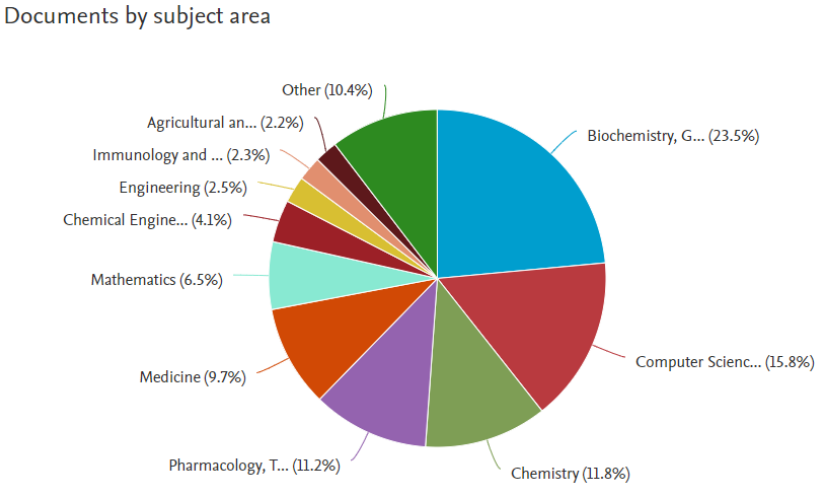


Figure 2. Documents by author and by subject area for results of "Prediction of drug-target affinity" in Scopus [11] from 2001 to 2023.

Initially, empirical methods such as surface plasmon resonance (SPR) and isothermal titration calorimetry (ITC) were widely used for experimental determination of binding affinities. However, these methods are time-consuming and resource-intensive [12]. On the other hand, high-throughput screening (HTS) techniques, though faster, often suffer from high false positive rates and require large libraries of compounds [13].

To address these limitations, early computational approaches like molecular docking were employed. Tools such as AutoDock and Glide simulated the interaction between a ligand and its target protein to predict binding affinity [14][15]. Despite their widespread use, these methods are computationally intensive and often struggle with accurately predicting affinities for large datasets or highly flexible molecules [16].

To overcome these challenges, Quantitative Structure-Activity Relationship (QSAR) models were developed. These models establish relationships between chemical structures and their biological activities using statistical techniques [17]. By utilizing molecular descriptors, QSAR models can predict the affinity of new compounds based on their structural similarities to known drugs [18]. However, QSAR models heavily depend on the quality of available experimental data and often require extensive feature engineering, which can lead to overfitting [19].

As time passed, advancements in machine learning (ML) and deep learning (DL) brought significant improvements in DTA prediction. These computational models can process large-scale biological and chemical data, capturing complex patterns and interactions that traditional methods might miss [20]. Kernel-based approaches, such as Support Vector Machines (SVM) and Random Forests, utilize predefined molecular descriptors to model DTA. Algorithms like KronRLS and SimBoost have shown promising results by mapping high-dimensional molecular data into a feature space where linear separation becomes feasible [21]. However, these methods are limited by their dependence on the selection of appropriate descriptors, which requires domain expertise and can be time-consuming [22].

Furthermore, matrix factorization techniques like Neighborhood Regularized Logistic Matrix Factorization (NRLMF) and Collaborative Matrix Factorization (CMF) decompose the interaction matrix into lower-dimensional representations, facilitating the prediction of unknown drug-target interactions [23]. Although effective in handling sparse interaction matrices, they often fail to capture non-linear relationships inherent in biological data [24].

It is observed that deep learning models have demonstrated significant improvements over traditional ML methods by learning hierarchical representations of data directly from raw inputs, bypassing the need for manual feature extraction [25]. For example, Convolutional Neural Networks (CNNs) have been employed to predict DTA by extracting features from molecular graphs or sequences. Models like DeepDTA and WideDTA leverage the spatial dependencies within molecular structures, achieving superior performance compared to traditional QSAR methods [26]. Despite their accuracy, CNNs are computationally intensive and require substantial amounts of data to train effectively [27].

In addition, Graph Neural Networks (GNNs) represent molecules as graphs, with atoms as nodes and bonds as edges. This architecture is particularly suited for capturing the topological features of molecules. Models such as GraphDTA uses different GNN architectures, depending on which on is selected, including Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), Graph Isomorphism Networks (GIN), and hybrid models like GAT-GCN, to directly model the molecular structure [28][29]. While GNNs excel in this domain, their performance can be sensitive to the size and connectivity of the molecular graphs [30].

Similarly, Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) and Transformer models, have been used to model sequential data such as protein sequences. These models capture long-term dependencies within sequences, making them suitable for DTA prediction [31]. However, RNNs can face challenges with very long sequences and require sophisticated techniques to mitigate issues like the vanishing gradient problem [32].

As mentioned, hybrid models combine multiple deep learning architectures, which exploit their individual strengths to enhance predictive accuracy. For instance, GAT-GCN models integrate Graph Attention Networks (GAT) with Graph Convolutional Networks (GCN),

capturing diverse features and leading to improved performance, though at the cost of increased complexity and computational requirements [33].

Additionally, to ensure the reliability and comparability of DTA prediction models, standardized benchmarking datasets and evaluation metrics are essential. Metrics such as Root Mean Square Error (RMSE), Mean Squared Error (MSE), Pearson correlation, Spearman correlation, and Concordance Index (CI) are commonly used to assess model performance [34]. Moreover, initiatives to establish and use standardized datasets, such as those from the DeepDTA and GraphDTA studies, facilitate fair comparison and reproducibility of results [35].

Looking forward, the future of DTA prediction lies in the integration of multi-omics data, enhancing model interpretability, and improving scalability. By combining several areas of omics data with molecular structures, models can achieve greater accuracy. Efforts to make models more interpretable will help in understanding the underlying biological mechanisms driving the predictions [36]. Additionally, optimizing computational efficiency and leveraging transfer learning can further advance the field [37].

In conclusion, the field of DTA prediction represents a significant advancement in computational drug discovery, with deep learning methods, particularly GNNs and hybrid models, leading the way. These models offer exceptional accuracy and the ability to capture intricate molecular interactions, significantly advancing the capability to predict drug-target affinities. As the field continues to evolve, addressing challenges related to scalability, interpretability, and data integration will be crucial for further progress.

Among the models mentioned throughout the study carried out on the evolution and emergence of DTA prediction techniques, GraphDTA method stands out among these advanced approaches for predicting drug-target binding affinity. GraphDTA leverages multiple Graph Neural Network architectures, as mentioned previously, including Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), Graph Isomorphism Networks (GIN), and GAT-GCN, to model molecular structures comprehensively. By doing so, GraphDTA captures the intricate topological features of molecules, enabling more accurate predictions of drug-target interactions. In the subsequent sections, we will delve deeper into the specifics of the GraphDTA method, exploring its architecture, performance, and the advantages it brings to DTA prediction.

3 Processing for "in-silico" drug-target affinity prediction methods

The main focus of this thesis is the GraphDTA model for the "in-silico" prediction of drug-target affinity. In order to conduct a comprehensive analysis of this prediction model, we will compare its architecture, performance, and other aspects with two other models: DeepDTA and WideDTA. GraphDTA focuses its structure on graph neural networks (GNNs), while DeepDTA focuses on convolutional neural networks (CNNs) with sequence-based encoding, and WideDTA combines CNNs with additional wide fully connected layers. Thus, all three are used for "in-silico" prediction but are based on different neural network architectures and distinct approaches for inputting the proteins and compounds data.

Before delving deeper into the structure of each model and their results for the same datasets, KIBA and Davis, we identified that all three models share common steps in pre-processing and processing within the workflow to predict drug-target affinities.

In the following subsections, we will analyze these two phases. This will help us understand that these "in-silico" DTA models follow a specific shared structure to achieve their results. While each model has its unique functions and classes that define it, they are "related" by the similar foundational structure in their architecture.

3.1.1 Pre-processing

This phase involves data loading, filtering, encoding, splitting, and pairing ligands with proteins. The key difference lies in how each model encodes SMILES strings: graphs for GraphDTA, and one-hot encoding or embeddings for DeepDTA and WideDTA.

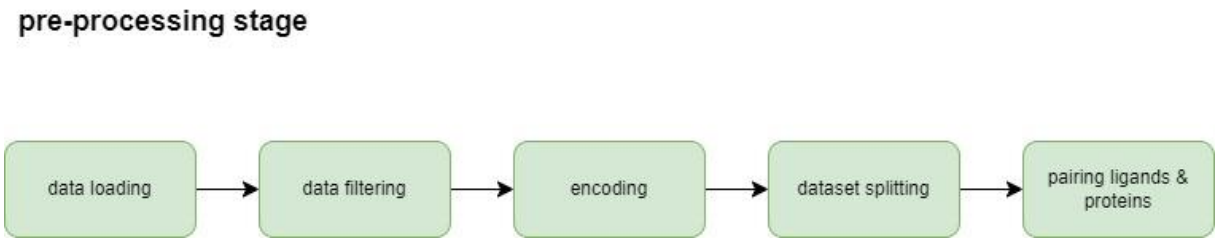


Figure 3. Workflow common to the three models (GraphDTA, DeepDTA and WideDTA) in the pre-processing stage [26, 28 and 41]. Diagram created with draw.io [39].

Data loading:

- SMILES strings (drugs): All models begin by loading SMILES strings that represent the chemical structure of drugs.
- Protein sequences: Protein sequences are also loaded for the targets.

Data filtering:

- SMILES and proteins length constraints: All models apply filtering to retain sequences within a specified length range (e.g., SMILES with fewer than 50 characters, proteins with fewer than 1000 characters).
- Handling missing data: Incomplete or erroneous data entries are filtered out to ensure a clean dataset for model training.

Encoding:

- SMILES strings: GraphDTA converts SMILES strings into graph representations (nodes for atoms and edges for bonds), while DeepDTA and WideDTA convert SMILES strings into one-hot encoded matrices or embeddings.
- Protein sequences:
  - One-hot encoding: All three models convert protein sequences into numerical matrices where each amino acid is represented by a unique binary vector.
  - Embedding layers (DeepDTA and WideDTA): Use embedding layers to transform these one-hot encoded sequences into dense feature vectors.

Dataset splitting:

- Train-test split: The dataset is split into training and testing subsets, using an 80/20 or 90/10 split ratio, ensuring that the model is evaluated on unseen data.

Pairing ligands and proteins:

- Ligand-protein pairing: All models create at least pairs of drug compounds (ligands) and protein sequences, each associated with a binding affinity value, which is the basis for the prediction tasks.

3.1.2 Processing

This phase focuses on feature extraction using CNNs in DeepDTA and WideDTA and GNNs in GraphDTA. All models use pooling, regularization, and fully connected layers to integrate features and predict binding affinities.

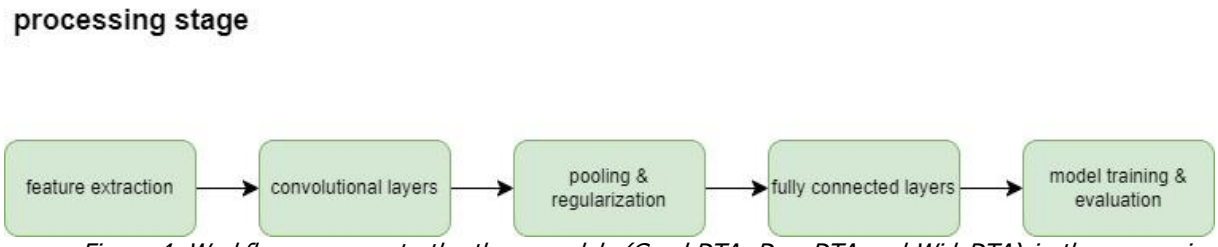


Figure 4. Workflow common to the three models (GraphDTA, DeepDTA and WideDTA) in the processing stage [26, 28, 38, 41-42]. Diagram created with draw.io [39].

Feature extraction:

- GraphDTA: Processes SMILES strings as graphs using Graph Neural Networks (GNNs), capturing structural relationships between atoms and bonds.
- DeepDTA and WideDTA: Use Convolutional Neural Networks (CNNs) to extract spatial and sequential features from one-hot encoded or embedded representations of SMILES and protein sequences.

Convolutional layers:

- SMILES and protein processing (DeepDTA and WideDTA): Employ 1D CNNs to capture local patterns and structural motifs in the SMILES strings and protein sequences.
- Graph convolutions (GraphDTA): Utilize graph convolutions to process the graph representations of drugs, learning the molecular substructures relevant to binding affinity.

Pooling and regularization:

- Max Pooling: Applied by all models to reduce the dimensionality of the extracted feature maps while retaining the most salient features.
- Dropout: All models incorporate dropout layer to prevent overfitting during training.

Fully connected layers:

- Feature integration: All models combine the extracted features from drugs and proteins through fully connected layers, creating a unified representation that captures the interaction potential.
- Prediction output: The combined features are passed through the final layers to output a continuous value representing the predicted binding affinity.

Model training and evaluation:

- Loss calculation: The models calculate loss (e.g., Mean Squared Error, RMS Loss) during training to measure the discrepancy between predicted and actual binding affinities.
- Optimization: Parameters are updated using optimizers like Adam to minimize the loss.
- Model evaluation: Post-training, the models are evaluated on the test set using metrics such as Mean Squared Error (MSE) or Concordance Index (CI) to assess prediction accuracy.

4 Comparative analysis of "in-silico" binding affinity prediction methods

4.1 Datasets (Kiba, Davis)

For the comparative analysis of "in-silico" drug-target affinity (DTA) prediction methods, two widely used datasets, KIBA and Davis, will be employed. These datasets are chosen due to their relevance and structure, which make them ideal for this type of prediction. They provide essential data such as binding affinities, drugs, targets, etc. [40-42]. This structure is crucial for forming accurate predictions and comparisons.

Moreover, the three models under study base their results on these datasets, making them integral to a fair comparison. By adhering to the methodologies prescribed by each model and using the same data types, we aim to ensure that the results obtained are comparable with those reported in the original studies. The analysis will not only explore the procedures and workflows of each model but also compare the results with those published in existing literature.

It is also noteworthy that the DeepDTA and WideDTA models, developed by MolPro, were initially created for the KIBA dataset. To extend their applicability, we performed the necessary preprocessing steps, such as motif creation, to test these models on the Davis dataset as well.

4.1.1 KIBA Dataset

The KIBA (Kinase Inhibitor BioActivity) dataset is a widely recognized resource for studying drug-target interactions, particularly involving kinase inhibitors. Kinases, which are enzymes that play critical roles in cell signaling and regulation, are key targets in drug discovery, especially for diseases like cancer. The KIBA dataset aggregates data from several bioactivity sources, including BindingDB, ChEMBL, and KinaseSARfari, and synthesizes various bioactivity measures (such as IC50, Kd, and Ki) into a single, unified KIBA score. This score represents the binding affinity between a kinase and its inhibitor, offering a comprehensive view of kinase inhibitor bioactivity.

The KIBA dataset is extensively used as a benchmark in computational drug discovery, particularly in machine learning models that predict drug-target interactions [43]. It includes detailed binding affinity information for 2,116 drugs and 229 targets, with scores ranging from 0.0 to 17.0. A score of 0.0 reflects the weakest binding affinity, suggesting minimal or no interaction, while a score of 17.0 indicates the strongest binding affinity, denoting a very strong interaction [28].

4.1.2 Davis Dataset

The Davis dataset is another key resource in drug-target interaction prediction, focusing specifically on binding affinity data between kinase inhibitors and kinase proteins. Unlike the KIBA dataset, which combines various bioactivity measures into a single score, the Davis dataset provides precise Kd values, which directly represent the dissociation constants of drug-target complexes.

The Davis dataset is highly regarded for its experimentally validated, high-quality binding affinities, making it an essential benchmark for evaluating computational models that predict drug-target interactions [44]. It includes binding affinity data for all combinations of 72 drugs

and 442 targets, with Kd constants ranging from 5.0 to 10.8. Lower Kd values indicate stronger binding affinities, reflecting a tighter interaction between the drug and its target [28].

4.2 GraphDTA

4.2.1 Introduction to key concepts

- **Graph regression**

Graph regression is a type of regression analysis where data is represented in the form of a graph. The objective in graph regression is to predict a continuous value associated with the nodes or edges of the graph. This method leverages the inherent structure of the graph, capturing relationships and interactions between nodes—such as atoms in molecular graphs—to enhance prediction accuracy. By considering the entire graph's structure, graph regression offers more context-aware and precise predictions compared to traditional regression methods [45].

- **Graph neural networks**

Graph Neural Networks (GNNs) are a class of neural networks specifically designed to work with graph-structured data. GNNs utilize the connections and interactions between nodes to learn meaningful representations of the graph. These networks can be applied to a range of tasks, including node classification, graph classification, and link prediction. The core concept involves iteratively aggregating information from a node's neighbors to update its representation, thereby capturing both local and global structures within the graph [46, 47].

- **Graph convolutional networks**

Graph Convolutional Networks (GCNs) are a specific type of GNN that extends the concept of convolutional neural networks (CNNs) to graph data. GCNs perform convolution operations on graphs, where each node aggregates information from its neighboring nodes to update its representation. This approach allows GCNs to effectively capture dependencies and interactions between nodes, making them well-suited for tasks involving graph data, such as predicting molecular properties or interactions in drug discovery [48, 49].

4.2.2 GraphDTA model

GraphDTA is a deep learning model specifically designed for predicting drug-target affinity (DTA). It represents drugs as molecular graphs, which allows the model to directly capture the intricate bonds between atoms, offering a more natural and informative representation compared to traditional methods that rely on one-dimensional (1D) sequences for drugs and proteins. The primary purpose of GraphDTA is to enhance the accuracy and meaningfulness of drug-target interaction predictions. By framing the prediction task as a regression problem, GraphDTA takes a drug-target pair as input and outputs a continuous value representing the binding affinity between the drug and the target. This approach facilitates more precise predictions, which are crucial in drug discovery and development.

GraphDTA combines the strengths of Graph Neural Networks (GNNs) and Convolutional Neural Networks (CNNs) to process and learn from both drug and protein data. Specifically, it

utilizes Graph Convolutional Networks (GCNs) to process the molecular graph of drugs and traditional CNNs to handle the sequence data of proteins. This hybrid approach enables the model to capture both the local and global structures of the input data, leading to improved prediction performance.

GraphDTA operates by representing drugs as molecular graphs and proteins as sequences of amino acids. The drug and protein representations are processed separately through specialized neural network architectures before being combined to predict the binding affinity.

For drugs, GraphDTA converts SMILES (Simplified Molecular Input Line Entry System) codes into molecular graphs using the chemical informatics software RDKit. Atom symbol, degree of the atom, number of hydrogens, implicit valence (number of bonds the atom can form based on its type), and whether the atom is part of an aromatic structure. The model employs one of four GNN variants to learn the graph representation:

1. GCN (Graph Convolutional Network) [48]
2. GAT (Graph Attention Network) [50]
3. GIN (Graph Isomorphism Network) [51]
4. A hybrid GAT-GCN architecture

These networks interpret drug compounds as graphs of atomic interactions, allowing GraphDTA to effectively capture the relationships between atoms within a molecule.

For proteins, GraphDTA uses a one-hot encoding method to manage the complexities of protein structures. Protein sequences are retrieved from the UniProt database using the gene name of the target. These sequences are composed of amino acids, each represented by a unique integer based on its alphabetical symbol. The sequences are either truncated or padded to a fixed length of 1,000 residues to maintain consistency, with shorter sequences padded with zeros. These integer sequences are then passed through embedding layers, which produce a 128-dimensional vector representation of the protein. The embedded sequences are processed using three 1D CNN layers to extract different levels of abstract features, followed by a max pooling layer to generate a compact representation vector of the protein sequence.

After obtaining the representations of both drugs and proteins, GraphDTA combines these features and passes them through several fully connected layers. This process ultimately predicts the binding affinity of the drug-target pair as the final output [28].

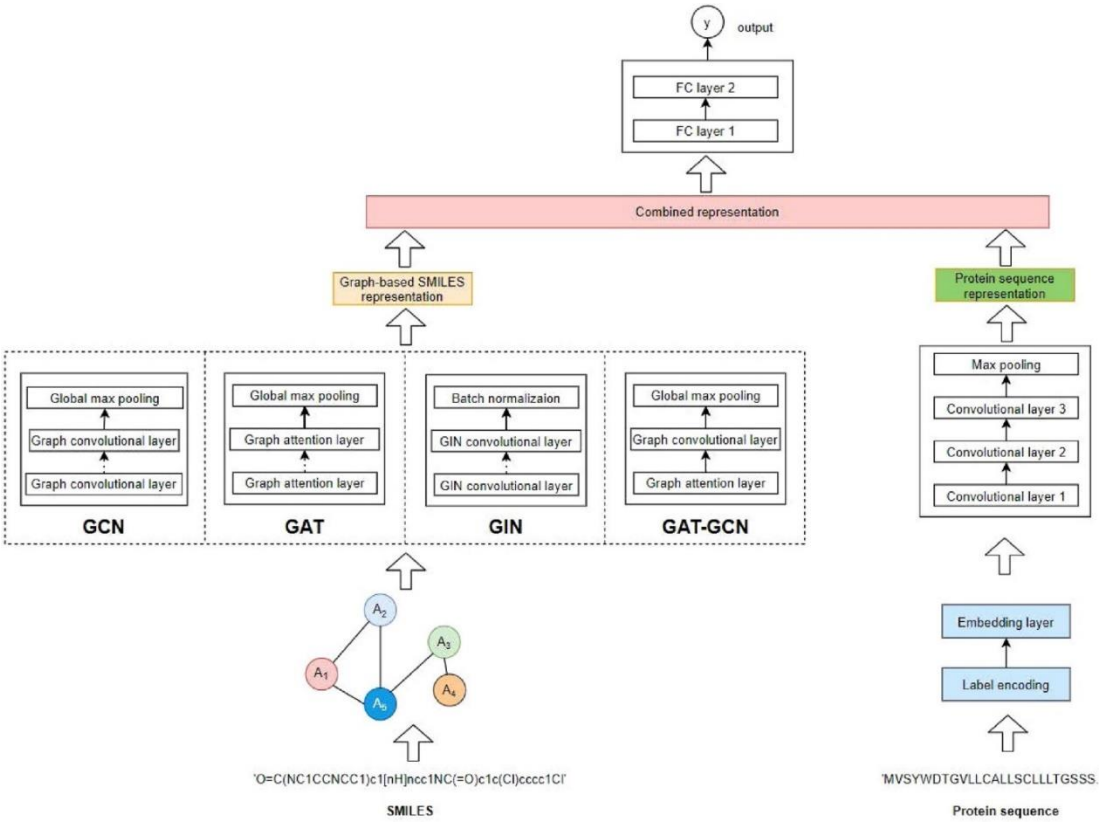


Figure 5. Workflow of GraphDTA for Predicting Drug-Target Binding Affinity [28].

4.2.3 Workflow

In this subsection, we will examine the functionality of each Python file provided by the GraphDTA model. Upon downloading the GitHub package associated with the GraphDTA study [42], we find four key Python scripts:

- **utils.py**

This module is a critical component of the drug-target affinity prediction pipeline. It is designed for preprocessing and managing datasets, converting molecular representations into graph structures, and computing various evaluation metrics essential for assessing model performance. The implementation leverages PyTorch and PyTorch Geometric libraries, which provide robust tools for handling graph-based data and constructing neural network models.

utils.py

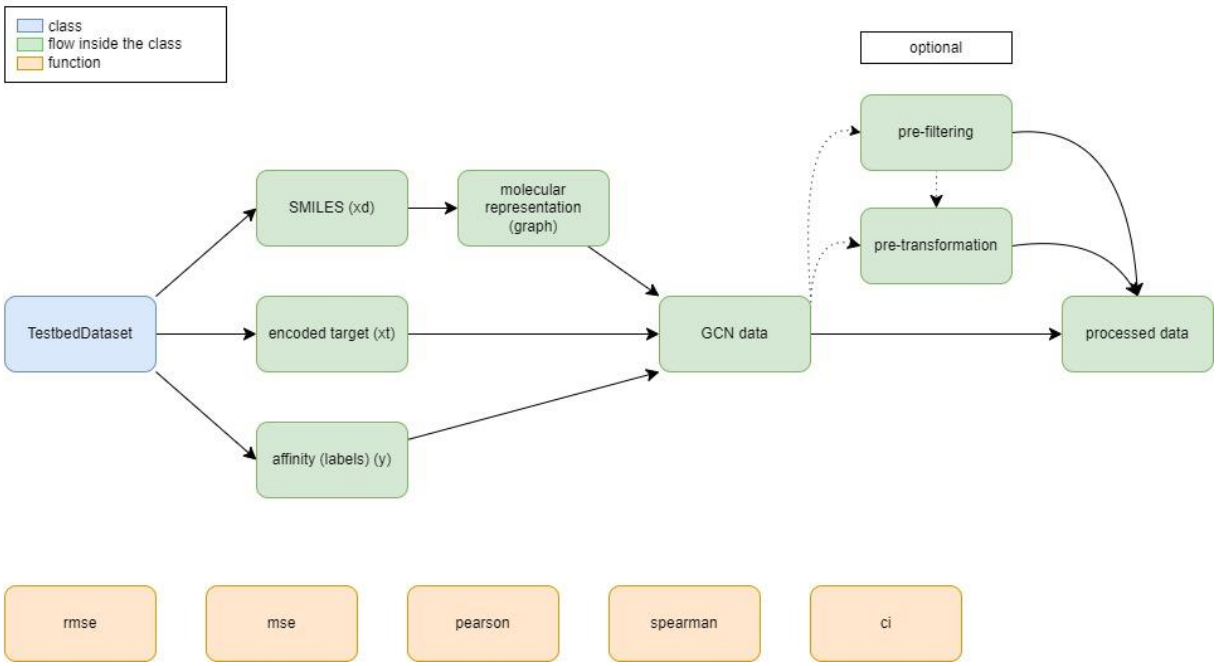


Figure 6. Diagram of utils.py, illustrating its class, workflow, and functions. Diagram created with draw.io [39].

- **TestbedDataset class**: This class is a custom extension of the InMemoryDataset class from the PyTorch Geometric library, specifically tailored for handling datasets in the context of drug-target affinity prediction. It streamlines the preprocessing, storage, and retrieval of data, ensuring an efficient computational workflow. The default input for this class is the Davis dataset.

  The process method, customized for drug-target affinity prediction, performs the following steps:

  · Input Validation: Ensures that the lengths of the input lists xd (for drugs/ligands), xt (for targets/proteins), and y (affinity) are equal.

  · Graph Construction: Converts each SMILES string (representing a molecule) into a graph representation using the smile_graph dictionary, which contains precomputed graph information. This involves extracting atomic features and constructing edge indices to represent the molecular structure.

  · Data Object Creation: Creates GCNData objects for each molecule-target pair, encapsulating the graph data and corresponding target sequence. These objects are appended to a data list.

  · Data Collation: Collates the data list into a format suitable for PyTorch Geometric and saves it to a file for future use.

  Additionally, there are optional steps for filtering unwanted data and applying any additional transformations.

The utils.py module also includes several utility functions for evaluating the performance of predictive models, which compute the following metrics:

- **rmse (Root Mean Square Error):** Calculates the root mean square error, a widely used metric for regression tasks, which measures the average magnitude of prediction errors. It is a widely used metric for regression tasks.
- **mse (Mean Square Error):** Computes the mean square error, providing a measure of the average squared difference between the predicted and actual values.
- **pearson (Pearson Correlation Coefficient):** Calculates the Pearson correlation coefficient, measuring the linear correlation between predicted and actual values, ranging from -1 to 1, with values closer to 1 indicating a strong positive correlation.
- **spearman (Spearman's Rank Correlation Coefficient):** Computes Spearman's rank correlation coefficient, assessing the monotonic relationship between predicted and actual values.
- **ci (Concordance Index):** Calculates the concordance index, which evaluates the ranking quality of predicted values by comparing the predicted and actual rankings—crucial for assessing model performance in ranking tasks.

- **create_data.py**

This script is pivotal in the preprocessing pipeline, transforming raw drug-target interaction data into formats suitable for graph-based deep learning models. Utilizing libraries such as RDKit for cheminformatics, NetworkX for graph creation, and Pandas for data manipulation, the script methodically processes data for subsequent modeling. It imports the utils.py script for preprocessing support.

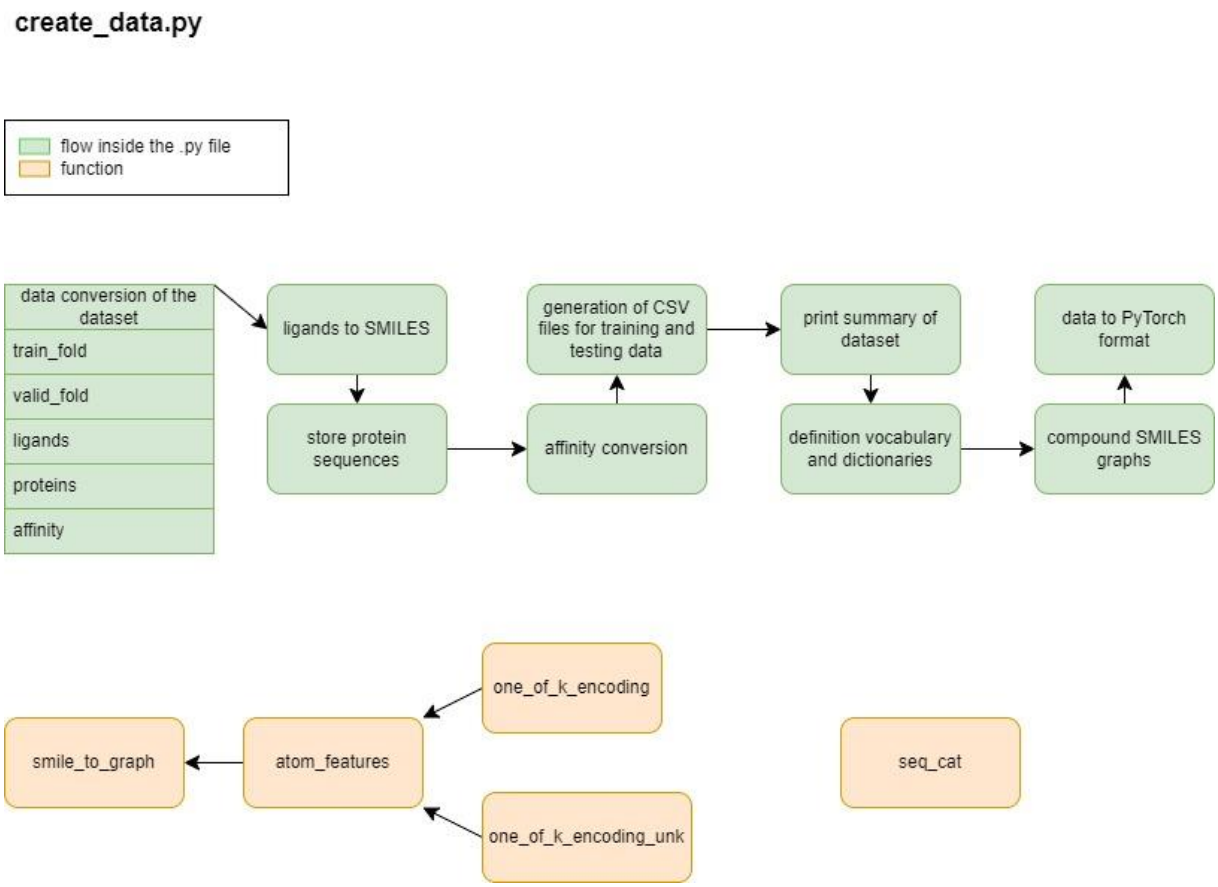


Figure 7. Diagram of create_data.py, showing its workflow and functions. Diagram created with draw.io [39].

Key functions include:

- **atom_features:** Encodes atom properties, such as symbol, degree, number of hydrogens, valence, and aromaticity, into a format suitable for graph-based models.
- **one_of_k_encoding:** Performs strict one-hot encoding for categorical features, ensuring that each input is represented as a unique binary vector corresponding to specified allowable categories.
- **one_of_k_encoding_unk:** Similar to one_of_k_encoding, but with added flexibility to handle unknown inputs by mapping them to a predefined category.
- **smile_to_graph:** Converts SMILES strings into graph representations using RDKit and NetworkX, extracting atomic features and bond indices to construct molecular graphs.
- **seq_cat:** Encodes protein sequences as numerical arrays based on a predefined vocabulary, enabling the representation of protein structures in a format compatible with computational models.

- **Workflow:**

  · Read and process DeepDTA dataset files: Converts raw datasets into a format suitable for graph neural network models, reading JSON files containing training/testing splits, ligand information, protein sequences, and affinity values.

  · Convert ligands to SMILES: Converts ligand structures to standardized canonical SMILES strings using RDKit, ensuring uniform molecular representation across the dataset.

  · Store protein sequences: Extracts protein sequences from the datasets and stores them in a list. This list is then used to encode the sequences into numerical representations, which are essential for deep learning models to process the protein information effectively.

  · Convert affinity values: It converts the affinity values only for the Davis dataset using a -log10 transformation, as the Davis dataset provides K_d values in nanomolar concentrations (nM), unlike the KIBA dataset which uses a different scale. This transformation helps to normalize the affinity values, making them more interpretable and numerically stable for model training.

  · Generate CSV files for training and testing data: Generates structured CSV files containing compound SMILES, target protein sequences, and affinity values. This step organizes the data into a structured format, making it easier to load and process during model training.

  · Print the summary of the dataset: Outputs a summary of the dataset, including the number of unique drugs and proteins, and the sizes of the training and testing splits, to give an overview of the data distribution and ensure proper data preparation.

  · Prepare vocabulary and dictionary for protein sequences: Defines a vocabulary of allowable amino acid characters and constructs a dictionary for encoding protein sequences numerically, ensuring that each amino acid is represented in a consistent format suitable for input into neural network models.

  · Generate graph representations of compounds: Constructs graph representations from SMILES strings, where each molecule is represented as a graph with nodes (atoms) and edges (bonds). These graphs are stored in a dictionary (smile_graph) for efficient retrieval and processing during model training.

· Convert data into PyTorch Geometric Data objects: Converts the processed data into PyTorch Geometric Data objects, saving them as .pt files for future training and evaluation.

- **training.py**

This script is central to the model training and evaluation process for predicting drug-target binding affinities using graph neural networks (GNNs). It orchestrates the entire training pipeline, from loading datasets to training models and evaluating their performance.

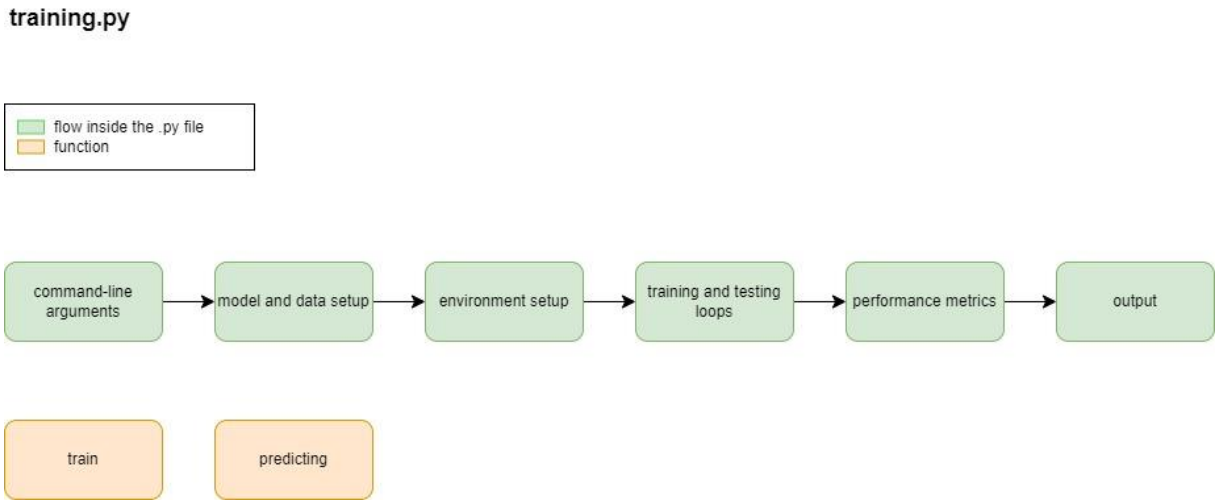


Figure 8. Diagram of training.py, illustrating its workflow and functions. Diagram created with draw.io [39].

Key functions include:

- **train:** Handles the training process of the model for a single epoch. It begins by printing the total number of samples that will be used for training, then sets the model to training mode to enable features like dropout. The function iterates over mini-batches of data provided by the train_loader, moving each batch to the specified computational device (such as a GPU). For each batch, it resets the gradients of the model's parameters, computes the model's output, and calculates the loss by comparing the output to the true labels. The gradients are then computed via backpropagation, and the optimizer updates the model's weights accordingly. To monitor training progress, the function logs information every few batches, including the current epoch number, the fraction of processed samples, and the current loss value. This structured approach allows for efficient and clear training across multiple epochs.
- **predicting:** Evaluates the model's performance on a test or validation dataset. It starts by switching the model to evaluation mode, which disables features that are only necessary during training. The function initializes empty tensors to store both the predictions and the true labels. Within a torch.no_grad() context, which disables gradient calculations for improved performance, the function processes each batch from the loader similarly to the training process but only focuses on obtaining the model's output. The predictions and corresponding labels for each batch are concatenated into larger tensors, enabling comprehensive evaluation after all batches have been processed. Finally, the function returns both the true labels and the

predicted values as flattened NumPy arrays, facilitating further analysis and performance metrics computation.

- **Workflow:**

  · Command-line arguments: Takes input via command-line arguments to specify the dataset (davis or kiba), modeling method (GINConvNet, GATNet, GAT_GCN, or GCNNet), and optionally, the CUDA device for training.
  · Model and data setup: Initializes the dataset and model based on command-line inputs, checking for the availability of processed data files. If not, it prompts the user to prepare the data using another script (create_data.py).
  · Environment setup: Defines key hyperparameters such as batch size, learning rate, logging interval, and the number of training epochs.
  · Training and testing loops: Iterates through the dataset, loading training and testing data, initializing the model and optimizer, and training the model over several epochs on the training dataset, evaluating performance at each step on the test dataset and saving the model if it achieves better performance.
  · Performance metrics: Computes various metrics (RMSE, MSE, Pearson correlation, Spearman correlation, and Concordance Index) to assess the model's predictive accuracy and reliability. The training function uses MSE as the loss function, with an Adam optimizer to update model parameters, to quantify the difference between predicted and actual binding affinities. An Adam optimizer is used to update the model parameters based on the computed gradients, ensuring efficient convergence.
  · Output: Saves results to a CSV file and the model state to a .model file when performance improvements are detected, ensuring the best model is preserved. The script keeps track of the model's performance ensuring that the best-performing model is preserved for future use.

- **training_validation.py**

This script enhances the model training and evaluation process by incorporating a validation step to prevent overfitting and ensure model robustness. It is critical for developing reliable GNN models for drug-target binding affinity prediction by monitoring model performance on a separate validation dataset during training, facilitating early stopping, and model selection based on validation metrics. This addition ensures that the model generalizes well to unseen data, enhancing its predictive reliability.

training_validation.py

Figure 9. Diagram of training_validation.py, showing its workflow and functions. Diagram created with draw.io [39].

Differences from training.py:

The core functions (train and predicting) and the workflows are similar to those in training.py. The key differences include:

- **Workflow:**
  · Training and testing loops: Splits the training data into training and validation subsets using an 80-20 split. This split ensures that a portion of the data is reserved for validation, which is crucial for monitoring the model's performance during training. Later, initializes the model and it creates PyTorch DataLoaders for training, validation, and testing, making them ready for mini-batch processing. The model is then trained using the training subset and evaluated on the validation data, allowing for real-time tracking of generalization capability and early stopping.
  · Performance metrics: In addition to the metrics calculated during training, the script checks if the MSE on the validation set is the best so far. If it is, the model's state is saved, and the model is further evaluated on the test data. If no improvement is observed, the script continues to the next epoch without saving the model.

4.2.4 Results

In this subsection, we present the results obtained using the GraphDTA model without modifying its original code. Our objective is to replicate the results intended by the authors by running the model on specified datasets to obtain Concordance Index (CI) and Mean Squared Error (MSE) values for each method (GCN, GAT_GCN, GAT, and GIN) used in predicting drug-target affinity.

Process overview:

To generate these results, the following steps were taken:

- **Data Preparation:** In the command window, the dataset was prepared by running 'python create_data.py'.
- **Model Training:** The prediction model was trained using the command python 'training.py 0 0 0', where the parameters correspond to the dataset type, the model, and the CUDA device respectively.
- **Model Training with Validation:** Alternatively, the model was trained with validation using 'python training_validation.py 0 0 0'. As before, '0' indicates the Davis dataset, '1' indicates the KIBA dataset, '0' represents GIN, '1' for GAT, '2' for GAT_GCN, and '3' for GCN, while the final '0' and '1' indicate the CUDA device.

Theoretical and reproduced results

Theoretical results for Davis dataset: GraphDTA

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.880	0.881	0.892	0.893
MSE	0.254	0.245	0.232	0.229

Table 1. Theoretical CI and MSE values for each method on the Davis dataset using GraphDTA (num of epochs = 1000, lr=0.0005).

Reproduced results for Davis dataset: GraphDTA

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.582	0.500	0.608	0.617
MSE	2.244	29.00	0.788	1.429

Table 2. Reproduced CI and MSE values for each method on the Davis dataset using GraphDTA (num of epochs = 3, lr = 0.003). Results obtained with the training_validation.py file.

Theoretical results for KIBA dataset: GraphDTA

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.889	0.891	0.866	0.882
MSE	0.139	0.139	0.179	0.147

Table 3. Theoretical CI and MSE values for each method on the KIBA dataset using GraphDTA (num of epochs = 1000, lr=0.0005).

Reproduced results for KIBA dataset: GraphDTA

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.583	0.535	0.638	0.628
MSE	0.686	0.811	0.628	0.663

Table 4. Reproduced CI and MSE values for each method on the KIBA dataset using GraphDTA (num of epochs = 3, lr = 0.003). Results obtained with the training_validation.py file.

This study evaluates the performance of various Graph Neural Networks (GNNs) within the GraphDTA framework—specifically, Graph Convolutional Network (GCN), Graph Attention Network (GAT), a combination of GAT with GCN (GAT_GCN), and Graph Isomorphism Network (GIN)—on two datasets: Davis and KIBA. The models are assessed using two key metrics: the Concordance Index (CI) and Mean Squared Error (MSE).

- Concordance Index (CI): This metric assesses the probability that a model correctly ranks the binding affinity for a randomly selected pair of drug-protein interactions. CI values range from 0 to 1, with 0.5 indicating random guessing and 1 indicating perfect prediction.
- Mean Squared Error (MSE): MSE measures the average squared difference between predicted and actual binding affinities. Lower MSE values indicate better predictive performance.

The results are divided into theoretical and reproduced categories. The theoretical results, drawn from the original GraphDTA study [7], were obtained under optimal conditions with extensive training (1000 epochs and a low learning rate of 0.0005). These results show high predictive accuracy with CI values above 0.880 and low MSE values. In contrast, the reproduced results were obtained using fewer training epochs (3) and a higher learning rate (0.003), due to the time constraints associated with model training. Although we successfully obtained results for the datasets using GraphDTA, these results are less reliable. The reduced number of epochs and higher learning rate led to significantly lower CI values and higher MSE values, particularly for the GAT_GCN method on the Davis dataset, which exhibited an anomalous MSE of 29.00. These discrepancies indicate that the models underperformed, likely due to underfitting caused by insufficient training.

For the Davis dataset, the theoretical results show that all models have high predictive accuracy, with CI values ranging from 0.880 (GCN) to 0.893 (GIN). MSE values are also low, ranging from 0.229 (GIN) to 0.254 (GCN), indicating strong predictive performance. These results were obtained under conditions of 1000 epochs and a learning rate of 0.0005, allowing extensive training of the models. Among these, the GIN model is the best-performing, with the highest CI of 0.893 and the lowest MSE of 0.229. On the other hand, the GCN model is the least effective, with the lowest CI of 0.880 and the highest MSE of 0.254. In contrast, the reproduced results show significantly lower CI values, ranging from 0.500 (GAT_GCN) to 0.617 (GIN), and much higher MSE values, particularly for GAT_GCN (29.00). This indicates reduced predictive accuracy and performance under the conditions used for reproduction, which included only 3 epochs and a learning rate of 0.003. These differing conditions likely caused the models to underfit and perform poorly. In this set of results, the GIN model still performs the good, with a CI of 0.617, while the GAT_GCN model is the worst, with a CI of 0.500 and an anomalous MSE of 29.00.

For the KIBA dataset, the theoretical results again show strong model performance, with CI values ranging from 0.866 (GAT) to 0.891 (GAT_GCN) and MSE values ranging from 0.139 (GCN and GAT_GCN) to 0.179 (GAT). As with the Davis dataset, these results were obtained under conditions of 1000 epochs and a learning rate of 0.0005. Among these models, the GAT_GCN model is the best-performing, with the highest CI of 0.891 and the lowest MSE of 0.139. Conversely, the GAT model is the least effective, with the lowest CI of 0.866 and the highest MSE of 0.179. The reproduced results for the KIBA dataset show lower CI values, ranging from 0.535 (GAT_GCN) to 0.638 (GAT), and higher MSE values, ranging from 0.628 (GAT) to 0.811 (GAT_GCN). These reproduced results, obtained under conditions of 3 epochs

and a learning rate of 0.003, suggest that the models were not trained as thoroughly, leading to poorer performance compared to the theoretical results. In this case, the GAT model emerges as the best, with a CI of 0.638, while the GAT_GCN model performs the worst, with a CI of 0.535 and an MSE of 0.811.

Key observations:

1. **Theoretical Performance:** The models exhibit slightly better performance on the Davis dataset in terms of CI, with the GIN model achieving the highest CI of 0.893. In contrast, the KIBA dataset shows slightly lower CI values, with the best-performing GAT_GCN model achieving a CI of 0.891. This suggests that the models generally perform well across both datasets under optimal conditions but show a marginal advantage on the Davis dataset.
2. **MSE Comparison:** MSE values are lower for the KIBA dataset under theoretical conditions, with the lowest being 0.139 for both GCN and GAT_GCN models. In contrast, the Davis dataset has slightly higher MSE values, with the lowest being 0.229 for the GIN model. This indicates that while CI values are slightly better for Davis, MSE suggests tighter predictive accuracy for the KIBA dataset.
3. **Impact of Reproduction:** Both datasets suffer a significant decline in performance under reproduced conditions, but the decline is more pronounced for the Davis dataset, particularly in the GAT_GCN model, which exhibits a dramatic increase in MSE to 29.00. The KIBA dataset, while also showing decreased performance, has more consistent results across the different models, with the worst MSE being 0.811 for the GAT_GCN model. This suggests that the Davis dataset may be more sensitive to changes in training conditions.
4. **Best and Worst Models:** For Davis dataset GIN and for KIBA GAT_GCN methods emerge as the best under theoretical conditions, where GAT_GCN performs the worst under reproduced conditions. This pattern highlights the importance of extensive training, as the GAT_GCN model may require more epochs or finer-tuned hyperparameters to achieve its full potential. While it is observed that the method GAT is the best for both models, which shows that suits more than the other methods for undertraining, but it is look to train correctly the datasets.

This analysis highlights the importance of training conditions in achieving reliable model performance and emphasizes the differing characteristics of the Davis and KIBA datasets. While both datasets show strong theoretical performance, the KIBA dataset appears more resilient to reduced training, whereas the Davis dataset is more sensitive, leading to greater discrepancies in the reproduced results. The theoretical results demonstrate the models' potential when adequately trained, while the reproduced results underscore the challenges of reproducibility in machine learning research, especially when time constraints necessitate reduced training. This underscores the critical need for appropriate hyperparameter tuning and sufficient training epochs to obtain meaningful predictive outcomes.

4.2.5 Discussion

The GraphDTA model represents a significant advancement in the field of drug-target affinity (DTA) prediction by leveraging graph-based deep learning approaches. Unlike traditional methods that often rely on one-dimensional sequences to represent drugs and proteins, GraphDTA uses a combination of Graph Neural Networks (GNNs) for drug molecules

(represented as molecular graphs) and Convolutional Neural Networks (CNNs) for protein sequences. This dual representation allows the model to capture both local interactions (e.g., atomic bonds within a molecule) and global structures (e.g., overall molecular shape and protein structure), which are crucial for accurately predicting the binding affinity between drugs and their target proteins.

The theoretical results provided in the original GraphDTA study demonstrate the high potential of this model, showing that with extensive training (1000 epochs and a learning rate of 0.0005), the model can achieve excellent predictive performance across different GNN variants. For example, on the Davis dataset, the GIN model achieved a CI of 0.893 and an MSE of 0.229, indicating strong predictive accuracy and low error.

However, when attempting to reproduce these results under less optimal conditions (fewer epochs and a higher learning rate due to time constraints), the performance significantly deteriorated. The CI values dropped considerably across all GNN variants, and the MSE values increased, with the most dramatic example being the GAT_GCN model on the Davis dataset, where the MSE shot up to 29.00—a clear indication of model underfitting. This discrepancy highlights the sensitivity of the model to training conditions and the importance of adequate training time and fine-tuning of hyperparameters for achieving optimal performance.

Analysis of performance on the Davis and KIBA Datasets:

- **Davis Dataset**: In the theoretical results, all models performed well, with CI values close to or above 0.880 and MSE values below 0.254. The GIN model emerged as the best performer, indicating that its architecture is particularly well-suited to capturing the nuances of drug-target interactions in this dataset. In contrast, the reproduced results revealed much lower CI values and higher MSE values, especially for the GAT_GCN model. This suggests that the model's complexity might require more extensive training to avoid underfitting and to fully leverage its potential.
- **KIBA Dataset**: Similarly, the theoretical results showed high CI values across all models, with the GAT_GCN model performing best (CI = 0.891, MSE = 0.139). However, the reproduced results again showed a significant drop in performance, particularly for the GAT and GCN models. This indicates that the model's effectiveness on the KIBA dataset also heavily depends on adequate training.

The stark contrast between the theoretical and reproduced results underscores several key points:

1. **Training Duration and Learning Rate**: The number of training epochs and the learning rate are critical in determining the performance of GraphDTA. The original study's extensive training allowed the models to converge to high-performing solutions, whereas the shortened training time in the reproduced experiments led to underfitting, where the models did not have enough time to learn from the data effectively, resulting in poor generalization to new data.
2. **Model Sensitivity**: Different GNN variants exhibit varying levels of sensitivity to training conditions. The GIN and GAT models, while generally performing well, still require fine-tuning to achieve optimal results. The GAT_GCN model's poor performance in the reproduced results suggests that it might be particularly prone to underfitting without sufficient training, where models may not realize their full potential leading to

inaccurate predictions that could have significant implications in drug discovery and development contexts.

To improve the reliability and performance of the GraphDTA model, the following approaches could be considered:

- **Increased Training Epochs**: Extending the number of training epochs, as done in the theoretical results, would likely enhance model performance, allowing it to better capture the complex patterns in the data.
- **Hyperparameter Tuning**: Further exploration of hyperparameters, such as learning rate, batch size, and model architecture, could help in finding a better balance between underfitting and overfitting.

In conclusion, while the GraphDTA model demonstrates high potential for drug-target affinity prediction, the results highlight the importance of comprehensive training and careful model tuning to achieve consistent and reliable outcomes.

4.3 DeepDTA

4.3.1 DeppDTA model

DeepDTA is a deep learning model specifically designed to predict drug-target interaction (DTI) strengths, focusing on the binding affinity between a drug and a protein target. Introduced by Öztürk et al. in 2018 [26], DeepDTA leverages convolutional neural networks (CNNs) to process both drug and protein sequences, which are represented as character-based sequences. The model's key innovation lies in its ability to directly utilize the SMILES representation of drugs and the amino acid sequences of proteins, enabling it to capture intricate local patterns that are crucial for understanding the binding interactions between a drug and its target.

The primary advantage of DeepDTA is its capacity to directly process SMILES strings and amino acid sequences as character sequences. This approach allows the model to identify and capture detailed local patterns within these sequences, which are essential for accurately predicting the binding interactions between drugs and their protein targets. By focusing on these detailed sequence patterns, DeepDTA significantly enhances the understanding of drug-target interactions, making it a valuable tool in drug discovery.

The DeepDTA model architecture involves processing drug and protein sequences separately through two distinct CNN blocks, each dedicated to one type of sequence—SMILES for drugs and amino acid sequences for proteins. Initially, these sequences are preprocessed, encoded, and then fed into their respective CNN blocks to extract relevant features. Each CNN block consists of three consecutive 1D-convolutional layers, with an increasing number of filters as the layers progress. This design allows the model to learn increasingly complex and abstract features from the sequences. Following the convolutional layers, max-pooling layers are applied to down-sample the output, aiding in generalization and dimensionality reduction.

In DeepDTA, drugs are represented by their SMILES (Simplified Molecular Input Line Entry System) strings—character sequences that describe the molecular structure. These SMILES strings are first integer/label encoded, assigning a unique integer to each character. For example, the model employs 64 unique labels for SMILES strings. The encoded sequences are then padded or truncated to a fixed maximum length based on the dataset used. These encoded sequences are subsequently transformed into dense vectors using Keras' Embedding layer, resulting in input matrices suitable for CNN processing.

Proteins in DeepDTA are represented by their amino acid sequences, which are similarly encoded as sequences of integers. The model uses 25 unique labels corresponding to the 20 standard amino acids and additional special characters. As with drug representation, these sequences are padded or truncated to fixed lengths. The encoded sequences are then converted into dense vectors via an embedding layer, ensuring compatibility with the CNN architecture.

After feature extraction from both the drug and protein sequences, these features are concatenated to form a combined feature set. This set is then passed through three fully connected (FC) layers. The first two FC layers consist of 1024 nodes each, while the final layer contains 512 nodes. Each of these layers is followed by dropout—a regularization technique that mitigates overfitting by randomly setting a fraction of activations to zero during training [26].

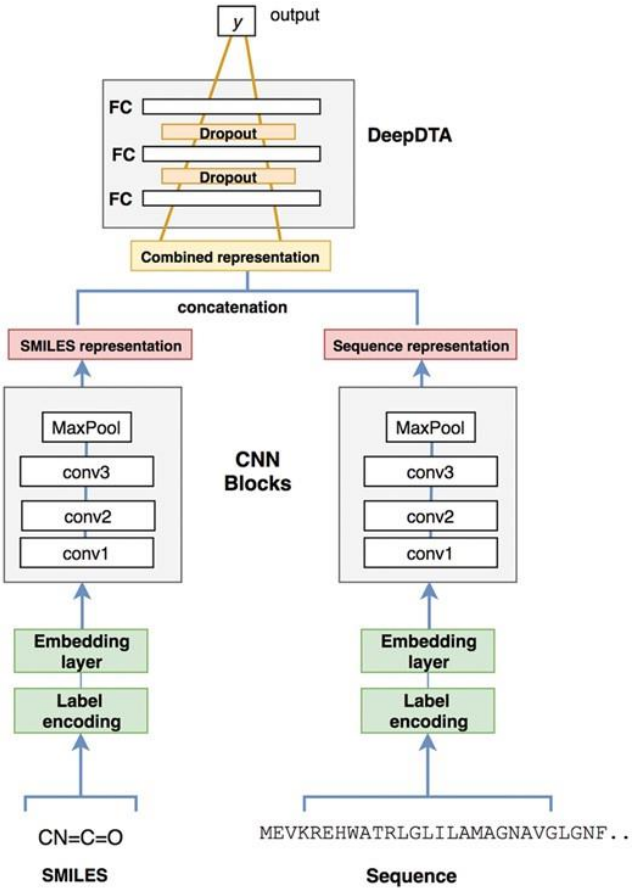


Figure 10. Workflow of DeepDTA for Predicting Drug-Target Binding Affinity [26].

4.3.2 Workflow

In this subsection, we examine how each Python file provided by MolPro [38] functions within the DeepDTA model, a deep learning architecture developed for predicting drug-target binding affinity. The model employs a character-based sequence representation approach, processing data from the KIBA dataset (Kinase Inhibitors Bioactivity data) where each protein and ligand has at least ten interactions. We further add to the code the path to accept the Davis dataset as an alternative input. Additionally, adjustments were made in the predict.py script to compute the Mean Squared Error (MSE) and Concordance Index (CI) values for affinity predictions, allowing for a direct comparison with results from Nguyen T.'s study [7].

The DeepDTA model architecture employs a 1D representation for both proteins and compounds. Protein sequences are one-hot encoded, where each amino acid is represented as a binary vector. Similarly, ligands are represented through SMILES strings, which are also one-hot encoded, converting each character in the SMILES string into a binary matrix.

Python files overview:

- **data.py**

The data.py module is crucial for data preprocessing in the DeepDTA model. It handles the loading, processing, and encoding of SMILES strings and protein sequences, and includes

a custom PyTorch Dataset class that structures the data for input into a Convolutional Neural Network (CNN).

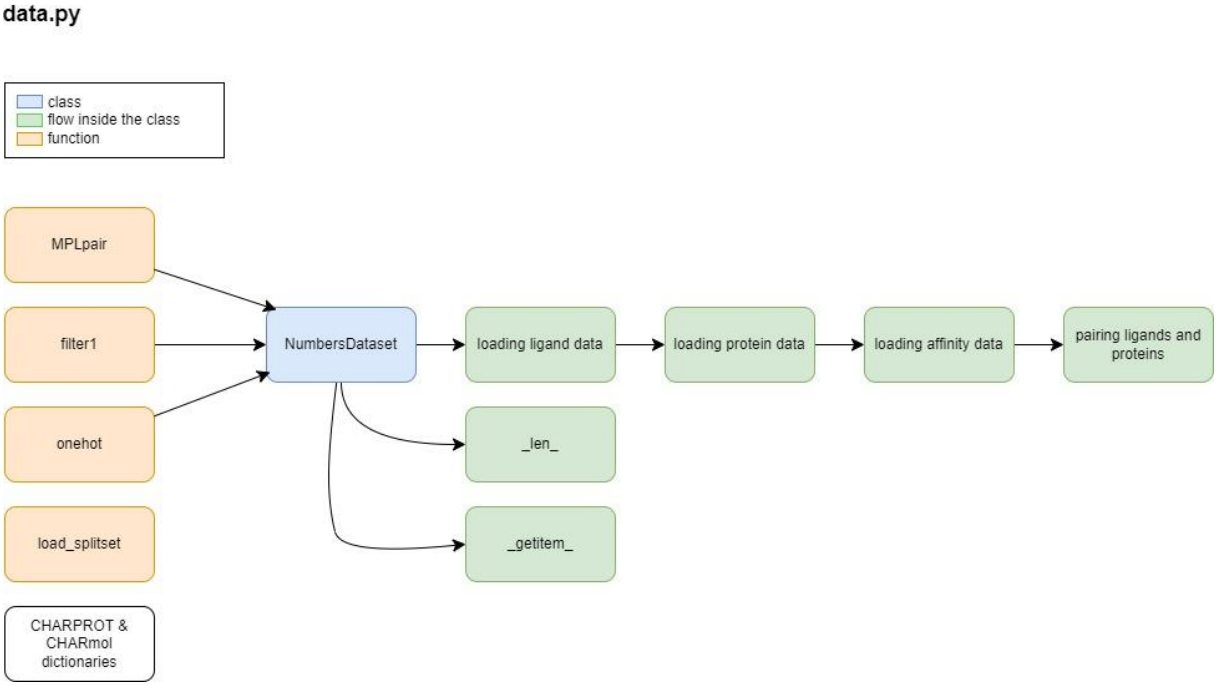


Figure 11. Diagram of data.py, illustrating its class, workflow, and functions. Diagram created with draw.io [39].

- **NumbersDataset class**: This class is a custom PyTorch Dataset class that organizes ligand, protein, and affinity data for input into the CNN model. It facilitates efficient data handling during training and evaluation by making the data accessible through PyTorch's DataLoader.

  The process method performs the following steps:

  · Loading ligand data: Ligand data, represented as SMILES strings, is loaded and filtered to retain sequences up to 50 characters, then one-hot encoded into a binary matrix suitable for neural network input.

  · Loading protein data: The protein sequences are loaded and because can be significantly longer than the ligand sequences, are filtered to include only those with lengths up to 600 characters. The filtered protein sequences are then one-hot encoded, transforming each amino acid in the sequence into a binary matrix.

  · Loading affinity data: The affinity matrix, which represents the binding affinity between various ligand-protein pairs, is loaded. This matrix is directly converted into a PyTorch tensor and stored, ensuring it is ready for use in model training and evaluation.

  · Pairing ligands and proteins: Ligand-protein pairs are formed along with their corresponding binding affinities. This pairing is performed by the MPLpair function, which systematically combines each one-hot encoded ligand with each one-hot encoded protein. The resulting pairs, along with their associated affinity values, are stored in the self.mol_pro list. This list serves as the core dataset that the model will use during training and testing.

· _len_: Provides a way to determine the size of the dataset by returning the length of the self.mol_pro list. This method is crucial for efficiently handling the dataset, especially when used with PyTorch's DataLoader, which requires knowledge of the dataset size to manage batching and iteration.

· _getitem_: Allows for indexed access to the dataset, enabling the retrieval of specific ligand-protein pairs and their binding affinities by their index. This method is integral to the DataLoader's ability to iterate over the dataset in a randomized or sequential manner, providing batches of data to the model during training or evaluation.

The data.py module also includes several key functions and dictionaries:

- **CHARPROT and CHARmol dictionaries:** Map unique characters in protein sequences and SMILES strings to integers for label encoding for model inputs. This encoding allows the sequences to be represented numerically for processing in neural networks. Also, facilitating the length of each dictionary.
- **MPLpair:** Pairs each ligand (SMILES string) with each protein sequence, creating a comprehensive dataset of input pairs and corresponding binding affinity values. This prepares the data for training by ensuring that each ligand-protein pair has a known affinity score.
- **filter1:** Filters protein sequences based on length, selecting only those below a specified threshold (l).
- **onehot:** Converts SMILES or protein sequences into one-hot encoded matrices based on their respective character sets. This function prepares the data for input into the CNN by converting each character in the sequences into a binary vector.
- **load_splitset:** Splits the dataset into training and testing sets. It creates PyTorch DataLoader objects that shuffle and batch the data for model training and evaluation.

- **model.py**

The model.py module defines the Convolutional Neural Network (CNN) architecture used in the DeepDTA framework, designed to effectively capture and combine local patterns in both the ligand and protein sequences, enabling accurate prediction of their binding affinity from encoded SMILES and protein sequences.

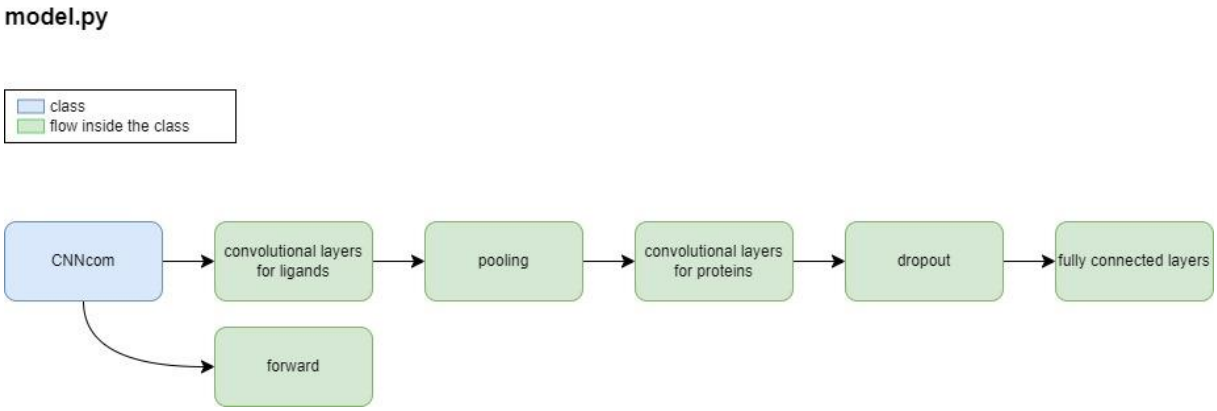


Figure 12. Diagram of model.py, illustrating its class and workflow. Diagram created with draw.io [39].

- **CNNcom class**: This class is a custom PyTorch neural network module that defines the CNN architecture, featuring separate convolutional layers for SMILES strings and protein sequences, followed by fully connected layers that merge features to predict binding affinity.

  The class performs the following steps:

  · Convolutional layers for ligands: The first convolutional layer takes the one-hot encoded ligand input and applies 16 filters with a kernel to capture local patterns in the ligand sequences. And the second convolutional layer further processes the output from the first layer, applying 32 filters. This deeper layer allows the network to capture more complex features from the ligand sequences.

  · Convolutional layers for proteins: Similarly, the first protein convolutional layer processes the one-hot encoded protein sequences with 16 filters. This layer extracts local features from the protein sequences. And the second protein convolutional layer applies 32 filters to the output of the first protein layer, enabling the network to learn more intricate patterns within the protein sequences.

  · Pooling and dropout: After the convolutional layer for ligands, a max-pooling operation is applied, reducing the spatial dimensions of the feature maps. This operation helps in reducing overfitting and improving the computational efficiency of the network. And a dropout layer is included after the first fully connected layer to further prevent overfitting by randomly setting a fraction of the input units to zero during training.

  · Fully connected layers: The output from the ligand and protein convolutional layers is flattened and concatenated into a single vector. This combined feature vector is passed through the first fully connected layer which reduces the dimensionality to 256 units while applying a ReLU activation function. And the final fully connected layer outputs a single value, representing the predicted binding affinity between the ligand and protein.

  · Forward method: Defines the data flow through the network. The ligand and protein inputs are passed through their respective convolutional and pooling layers, then flattened and concatenated. The concatenated vector is processed through the fully connected layers to produce the final affinity prediction.

- **train.py**

The train.py module manages the entire process of data preparation, model training, defining the loss function, selecting the optimizer, executing the training loop, and saving the trained model. This ensures an efficient workflow and allows the trained model to be reused or evaluated later.

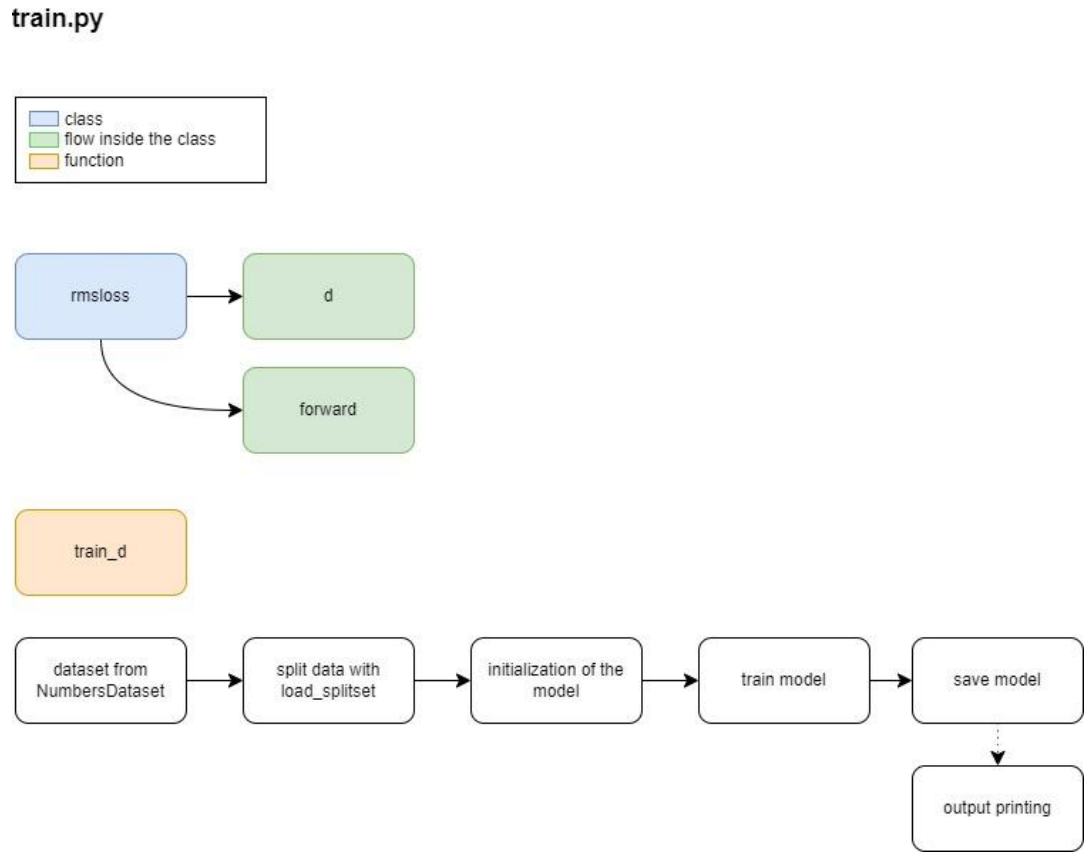


Figure 13. Diagram of train.py, illustrating its workflow and function. Diagram created with draw.io [39].

- **rmsloss class**: This class defines a custom loss function that computes the root mean square error (RMSE) between the predicted and actual binding affinities. It extends the nn.Module class from PyTorch enabling it to integrate seamlessly into the PyTorch ecosystem.

  The class incorporates:

  · Initialization: Initializes by creating an instance of the nn.MSELoss() function, which calculates the Mean Squared Error (MSE) between predictions and targets.

  · Forward: The MSE loss is computed first and then the square root of this value is returned to provide the RMSE. This method ensures that the loss metric directly corresponds to the scale of the target variable, making it more interpretable.

The train.py module includes the following function:

❖ **train_d:** Implements the training loop for the CNN model. It iteratively updates the model's parameters using backpropagation and the Adam optimizer. The function trains the model over a specified number of epochs, calculating the loss for each batch and updating the model weights accordingly. The trained model is then saved for later use.

- **Workflow:**

  · Dataset preparation: The dataset is created using the NumbersDataset class, which handles loading and preprocessing of data related to ligand, protein, and affinity information. This class is pivotal for structuring the data in a format suitable for model input.
  · Data splitting: Once the dataset is prepared, it is split into training and testing sets with a split ratio of 0.2 in this case, meaning 20% of the data is reserved for testing, and returns two data loaders: train_loader for the training set and test_loader for the testing set. This step is essential for ensuring that the model is trained on one portion of the data while being evaluated on another to prevent overfitting.
  · Model initialization: The model2 object is instantiated from the CNNcom() class, which defines the convolutional neural network (CNN) architecture. This model will be trained to predict outcomes based on the input data.
  · Model training: The model is trained with train_loader providing the data. This function executes the training loop, optimizing the model parameters to minimize the RMSE loss. The output of this process is stored in the tt variable, which contains the lists of loss values and model outputs for each batch.
  · Model saving: After the training process is complete, the trained model (model2) is saved to a file named deep.pt. This allows the model to be reloaded for future use, such as further training, fine-tuning, or evaluation on new data.
  · Output printing: If the script is run directly (not imported as a module), it prints the loss and output lists to provide immediate feedback on the training process.

- **predict.py**

The predict.py module evaluates the performance of a trained model on test data by generating predictions and computing key evaluation metrics, such as Mean Squared Error (MSE) and Concordance Index (CI). It uses the calculate_metrics function to compute these metrics and the predict_and_evaluate function to manage the prediction process and integrate metric calculations. The main workflow in this module involves loading the model, preparing the data, and using these functions to comprehensively assess the model's performance, providing a complete and efficient evaluation process.

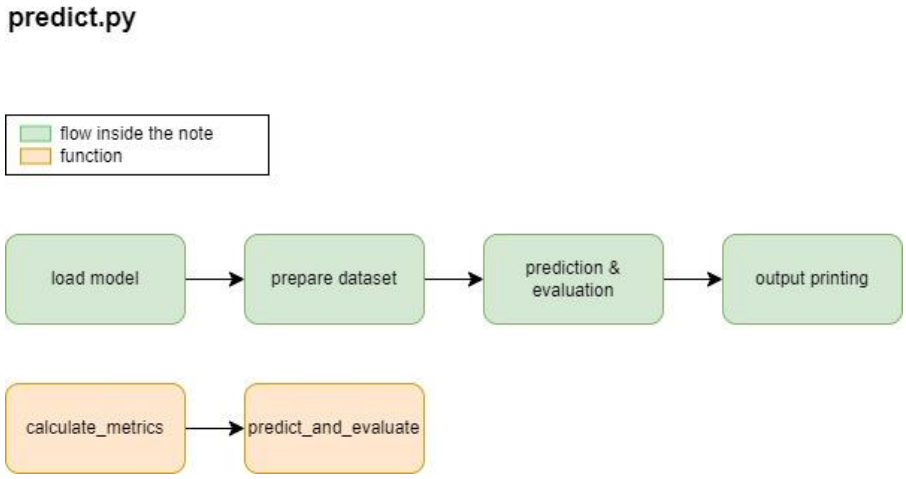


Figure 14. Diagram of predict.py, showing its workflow and functions. Diagram created with draw.io [39].

Functions:

- **calculate_metrics:** Computes the Mean Squared Error (MSE) and Concordance Index (CI) between the model's predictions and the actual target values. It first checks whether the predictions input is a list of tensors or a single tensor; if it's a list, the tensors are concatenated into one. The function then converts both predictions and actuals to NumPy arrays and calculates the MSE, which measures the average squared difference between predictions and targets, and the CI, which assesses the ranking quality of predictions. The function returns these two metrics, providing a quantitative evaluation of the model's performance.
- **predict_and_evaluate:** Generates predictions on a test dataset using a trained model and evaluates these predictions by calculating the Mean Squared Error (MSE) and Concordance Index (CI) metrics. The process involves iterating over the test data, reshaping inputs as necessary, and accumulating both the predictions and actual target values. After processing a specified number of batches, the function concatenates the predictions and actual values into single tensors. It then calls calculate_metrics to compute the MSE and CI, providing both the model's predictions and a quantitative measure of its accuracy and ranking performance.

- **Workflow:**

  · Model loading: The previously trained model is loaded (deep.pt). This model will be used for making predictions on the test dataset.
  · Dataset preparation: The NumbersDataset class is used to create the dataset, which is then split into training and test sets using load_splitset. In this case, 10% of the data is allocated for testing (.1 split ratio).
  · Prediction and Evaluation: The predict_and_evaluate function is called with the loaded model and test data. This function performs predictions on the test set and calculates the MSE and CI metrics.
  · Output printing: Finally, the predictions, MSE, and CI values are printed. This provides both the predicted outcomes and a quantitative evaluation of how well the model has performed on the test data.

4.3.3 Results

In this subsection, we present the results obtained using the DeepDTA model without any modifications to its original code. Our goal was to replicate the results reported by the original authors by running the model on the Davis and KIBA datasets. We focused on evaluating the Concordance Index (CI) and Mean Squared Error (MSE) of the model in predicting drug-target affinity.

Process overview:

The following steps were taken to generate the results:

- **Model Preparation:** The model can be prepared and trained by running the command 'python train.py'.
- **Model Prediction:** The prediction model was executed using 'python predict.py', where the paths were adjusted according to the dataset being used.

Theoretical results for Davis dataset: DeepDTA

Values\Model	DeepDTA
CI	0.878
MSE	0.261

Table 5. Theoretical CI and MSE values on the Davis dataset using DeepDTA (1D, 1D).

Reproduced results for Davis dataset: DeepDTA

Values\Model	DeepDTA
CI	7835510.0
MSE	0.456

Table 6. Reproduced CI and MSE values for the Davis dataset using DeepDTA (1D, 1D) (epochs = 3, lr = 0.003).

Theoretical results for KIBA dataset: DeepDTA

Values\Model	DeepDTA
CI	0.863
MSE	0.194

Table 7. Theoretical CI and MSE values on the KIBA dataset using DeepDTA (1D, 1D).

Reproduced results for KIBA dataset: DeepDTA

Values\Model	DeepDTA
CI	0.605
MSE	30.61

Table 8. Reproduced CI and MSE values for the KIBA dataset using DeepDTA (1D, 1D) (epochs = 3, lr = 0.003).

This study evaluates the performance of the DeepDTA model on the Davis and KIBA datasets, with a focus on two key metrics: the Concordance Index (CI) and Mean Squared Error (MSE).

The results are categorized into theoretical and reproduced values. The theoretical results, as reported in GraphDTA study [7], were obtained under optimal conditions with extensive training. These results demonstrate high predictive accuracy, with CI values close to 0.880 for the Davis dataset and 0.860 for the KIBA dataset. Correspondingly, the MSE values are also low, indicating strong predictive performance. In contrast, the reproduced results, obtained using only 3 epochs and a learning rate of 0.003, show significant discrepancies, particularly in the CI values. For the Davis dataset, the reproduced CI is 7835510.0, which is an anomalously high value and suggests a calculation or model error. However, the reproduced

MSE value of 0.456, while higher than the theoretical value, still indicates reasonable predictive accuracy. For the KIBA dataset, the reproduced CI of 0.605 is substantially lower than the theoretical value of 0.863, and the MSE of 30.61 is significantly higher, indicating poor model performance under the reproduction conditions.

For the Davis dataset, the theoretical results for the DeepDTA model show high predictive accuracy, with a CI of 0.878 and an MSE of 0.261. These results were obtained under conditions of extensive training, allowing the model to fully learn the complex patterns in the data. In contrast, the reproduced results show a significant anomaly in the CI value, which is reported as 7,835,510.0, clearly indicating an error in the reproduction process. The MSE, however, is 0.456, which is slightly higher than the theoretical value, suggesting some loss in predictive accuracy likely due to underfitting. The reproduction was performed under reduced conditions of 3 epochs and a learning rate of 0.003, which likely contributed to these discrepancies. The model was not originally designed for the Davis dataset, but was used for this analysis, which may have introduced additional challenges in the reproduction process.

For the KIBA dataset, the theoretical results for DeepDTA also demonstrate strong performance, with a CI of 0.863 and an MSE of 0.194. These metrics indicate that the model is effective at predicting drug-target affinities when trained under optimal conditions. However, the reproduced results show a considerable decline in performance, with the CI dropping to 0.605 and the MSE increasing dramatically to 30.61. This significant increase in MSE suggests that the model struggled to generalize under the reproduced conditions, while the lower CI reflects a poor ranking of the predictions relative to the actual binding affinities. This decline in performance is likely due to the reduced number of training epochs and the higher learning rate, which did not allow the model to adequately capture the underlying relationships in the data.

Key observations:

1. **Theoretical Performance:** The theoretical results indicate that the DeepDTA model performs well on both the Davis and KIBA datasets under optimal conditions, with CI values close to 0.880 and low MSE values, reflecting strong predictive accuracy.
2. **MSE Comparison:** The MSE values are higher in the reproduced results for both datasets, particularly for the KIBA dataset, where the MSE increases dramatically to 30.61. This suggests that the model underperformed due to the reduced number of training epochs and higher learning rate.
3. **Impact of Reproduction:** Both datasets show a significant decline in performance under reproduced conditions, with the Davis dataset showing an anomalous CI value and the KIBA dataset showing both a reduced CI and a dramatically increased MSE. This indicates that the DeepDTA model is sensitive to changes in training conditions, particularly the number of epochs and learning rate.
4. **Model Sensitivity:** The discrepancies between the theoretical and reproduced results highlight the importance of sufficient training and appropriate hyperparameter tuning. The KIBA dataset, in particular, appears to be more resilient to reduced training, but the dramatic increase in MSE underlines the need for careful consideration of model parameters.

These results highlight the importance of sufficient training and careful hyperparameter tuning when using the DeepDTA model. The large discrepancies between the theoretical and reproduced results, particularly in the KIBA dataset, emphasize the model's sensitivity to training conditions and the data handling workflow. Moreover, these findings underscore potential shortcomings in the model's training process when applied to different datasets. The theoretical results demonstrate the model's potential under ideal circumstances, but the reproduced results underscore the challenges of maintaining that performance in less rigorous training scenarios. This analysis illustrates the critical need for meticulous model adaptation and validation when applying machine learning models across varying datasets.

4.3.4 Discussion

The DeepDTA model represents an important development in the field of drug-target affinity (DTA) prediction by utilizing deep learning techniques to directly predict binding affinities from the sequences of drugs and proteins. Unlike traditional methods, which often require extensive feature engineering or rely on indirect indicators, DeepDTA employs Convolutional Neural Networks (CNNs) to automatically extract relevant features from raw sequence data, streamlining the prediction process and potentially improving accuracy.

The original DeepDTA study showcased the model's strong performance across various benchmarks, particularly with extensive training (e.g., 100 epochs and a learning rate of 0.001). The theoretical results demonstrated the model's ability to achieve high predictive accuracy. For example, on the Davis dataset, DeepDTA achieved a Concordance Index (CI) of 0.878 and a Mean Squared Error (MSE) of 0.261, indicating both strong ranking ability and low prediction error. Similarly, on the KIBA dataset, the model achieved a CI of 0.863 and an MSE of 0.194, reinforcing its robustness across different datasets.

However, when attempting to reproduce these results under less optimal conditions—specifically with a significantly reduced number of training epochs and a different learning rate—the model's performance deteriorated noticeably. For instance, in the reproduced results for the Davis dataset, the CI value was reported to be anomalously high at 7,835,510.0, which clearly points to a calculation error or an issue during reproduction. Even ignoring this anomaly, the MSE increased to 0.456, reflecting a notable decrease in predictive accuracy. On the KIBA dataset, the CI dropped to 0.605, and the MSE spiked to 30.61, further emphasizing the model's sensitivity to the training regimen.

Analysis of performance on the Davis and KIBA Datasets:

- **Davis Dataset:** The original results for the Davis dataset demonstrated strong performance, with CI values close to or above 0.878 and MSE values below 0.261. These metrics suggest that DeepDTA's architecture is particularly effective at capturing the intricate relationships between drugs and protein targets in this dataset. However, the reproduced results painted a different picture, with the model struggling to maintain its predictive accuracy under reduced training conditions. The high MSE and the anomalous CI value suggest underfitting, where the model did not have enough training time to properly learn from the data. Additionally, it should be noted that the code used in our reproduction was originally designed for the KIBA dataset, and we applied it to the Davis dataset without modifications. As a result, the code may not be well-suited for datasets other than KIBA.

- **KIBA Dataset:** The original study reported high CI values across all models on the KIBA dataset, with the DeepDTA model achieving a CI of 0.863 and an MSE of 0.194. These results demonstrate the model's capacity to generalize across different datasets with varied characteristics. However, in the reproduced experiments, the model's performance significantly declined, particularly with an increased MSE of 30.61. This drastic change indicates that the model's effectiveness is heavily dependent on adequate training, and the reduced training regimen was insufficient to capture the complex patterns in the KIBA dataset.

The stark contrast between the theoretical and reproduced results highlights several critical factors that influence the performance of the DeepDTA model:

1. **Training Duration and Learning Rate:** The number of training epochs and the learning rate are crucial determinants of the DeepDTA model's performance. In the original study, extensive training allowed the model to converge on highly effective solutions, whereas the shortened training time in the reproduced experiments led to underfitting. This underfitting occurred because the model did not have sufficient time to learn the complex relationships between drugs and proteins, resulting in poor generalization and increased prediction errors.
2. **Model Sensitivity:** DeepDTA, like many deep learning models, exhibits sensitivity to the training conditions, including hyperparameters like learning rate and batch size. The model's poor performance under the reproduced conditions suggests that it may be particularly prone to underfitting without sufficient training. This underfitting can lead to inaccurate predictions, which is especially concerning in drug discovery contexts where precision is critical.
3. **Reproducibility Challenges:** The significant discrepancies between the theoretical and reproduced results also underscore the challenges of reproducibility in machine learning research. Even slight deviations in training protocols or data preprocessing can lead to vastly different outcomes, highlighting the need for detailed and transparent reporting of experimental conditions in published studies.

The findings from this study suggest several important directions for improving the reliability and performance of the DeepDTA model:

- **Increased Training Epochs:** Extending the number of training epochs to match those used in the original study would likely enhance the model's ability to capture the complex patterns within the data, thereby improving predictive accuracy.
- **Hyperparameter Tuning:** Further exploration of hyperparameters, such as learning rate, batch size, and model architecture, could help strike a better balance between underfitting and overfitting, ensuring that the model generalizes well to new data.

In conclusion, the DeepDTA model demonstrates significant potential for drug-target affinity prediction, as evidenced by its strong theoretical performance on the Davis and KIBA datasets. However, the reproduced results reveal the model's sensitivity to training conditions, highlighting the importance of extensive training and careful hyperparameter tuning to achieve consistent and reliable outcomes. These findings emphasize the need for rigorous testing, validation, and reproducibility in future research, ensuring that deep learning models like DeepDTA can be effectively applied in real-world drug discovery scenarios.

4.4 WideDTA

4.4.1 WideDTA model

WideDTA is a deep learning model specifically designed to predict drug-target binding affinity (DTA). It integrates both wide and deep learning components to enhance the prediction accuracy of drug-target interactions. Unlike models that rely solely on either sequence-based or structure-based features, WideDTA combines various molecular descriptors and embeddings to capture a broader spectrum of information from both drugs and proteins. This hybrid approach allows WideDTA to effectively model complex interactions and provide more reliable predictions of binding affinities between drugs and their protein targets.

WideDTA is designed to advance computational drug discovery by accurately predicting the binding affinity between drugs and their target proteins. By effectively modeling the complex interactions between drugs and proteins, WideDTA can help in the identification of promising drug candidates and the optimization of drug design. Its ability to predict binding affinities more accurately than traditional methods makes it a valuable tool for researchers and pharmaceutical companies looking to accelerate the drug discovery process and reduce the costs associated with experimental testing.

WideDTA combines the strengths of convolutional neural networks (CNNs) and embedding techniques to process and learn from both drug and protein data. The model architecture involves multiple CNN blocks, each designed to handle different types of input data, such as drug compounds, protein sequences, protein domains and motifs, and molecular common substructures (MCS). Each CNN block contains two 1D convolutional layers, followed by a max pooling layer to down-sample the output. This design allows the model to capture both general and specific patterns in the input data, enhancing its ability to learn complex interactions.

WideDTA operates by processing multiple types of input data through specialized CNN blocks designed to handle the unique characteristics of each type. Each CNN block within the architecture is tuned to extract specific features from the embeddings, allowing the model to learn both simple and complex patterns in drug-target interactions. The model architecture features several 1D CNN blocks, each with two convolutional layer followed by a max pooling layer to then being combined to predict the binding affinity.

In WideDTA, drug representation is achieved through a combination of SMILES-based embeddings and additional chemical descriptors. SMILES strings, which provide a linear notation for representing chemical structures, are processed using embedding layers to convert them into continuous feature vectors. These embeddings are enhanced by incorporating molecular descriptors, such as molecular weight, logP, and topological surface area, which capture key physicochemical properties of the drug. This dual approach ensures that the model takes into account both the structural and physicochemical characteristics of the drug. The output from these embedding layers is then fed into the corresponding CNN block for further processing.

For protein representation, WideDTA utilizes sequence-based embeddings derived from protein sequences. The sequences are encoded numerically, with each amino acid represented by a unique integer. These sequences are then passed through embedding layers that generate continuous vectors, capturing the biochemical and structural information of the proteins. Additionally, WideDTA incorporates evolutionary information through Position-Specific Scoring Matrices (PSSMs), which highlight the conservation and importance of specific

residues within the protein sequence. This combination of sequence-based embeddings and evolutionary data allows the model to account for both the structural and functional aspects of proteins. The embedded sequences are subsequently processed through a dedicated CNN block.

For ligand molecular common substructure (MCS), which are used to provide additional information about the drug's potential interactions with proteins, they are also embedded into continuous vectors through a dedicated embedding layer. By incorporating MCS, the model can learn common patterns and pharmacophores—specific arrangements of atoms that are responsible for the biological activity of the molecules. The embedded MCS data is then processed through a designated CNN block to extract relevant features..

For protein motifs and domains, which provide essential information about the protein's functional and structural regions. Motifs are short, conserved regions within protein sequences that are often involved in specific functions or interactions, while domains are larger structural units within proteins that often fold independently and have distinct functions. These motifs and domains are encoded into numerical representations and then embedded using an additional embedding layer. The embeddings capture the functional significance of specific protein regions and their potential roles in drug binding. The embedded data is then processed through a separate CNN block to uncover complex patterns that could influence binding affinity.

After obtaining the representations from the four blocks, WideDTA concatenates the outputs to create a combined feature vector, which will go through a series of fully connected layers to further refine the predictions. It was also used dropout layers to help to prevent overfitting, ensuring that the model generalizes well to new data [41].

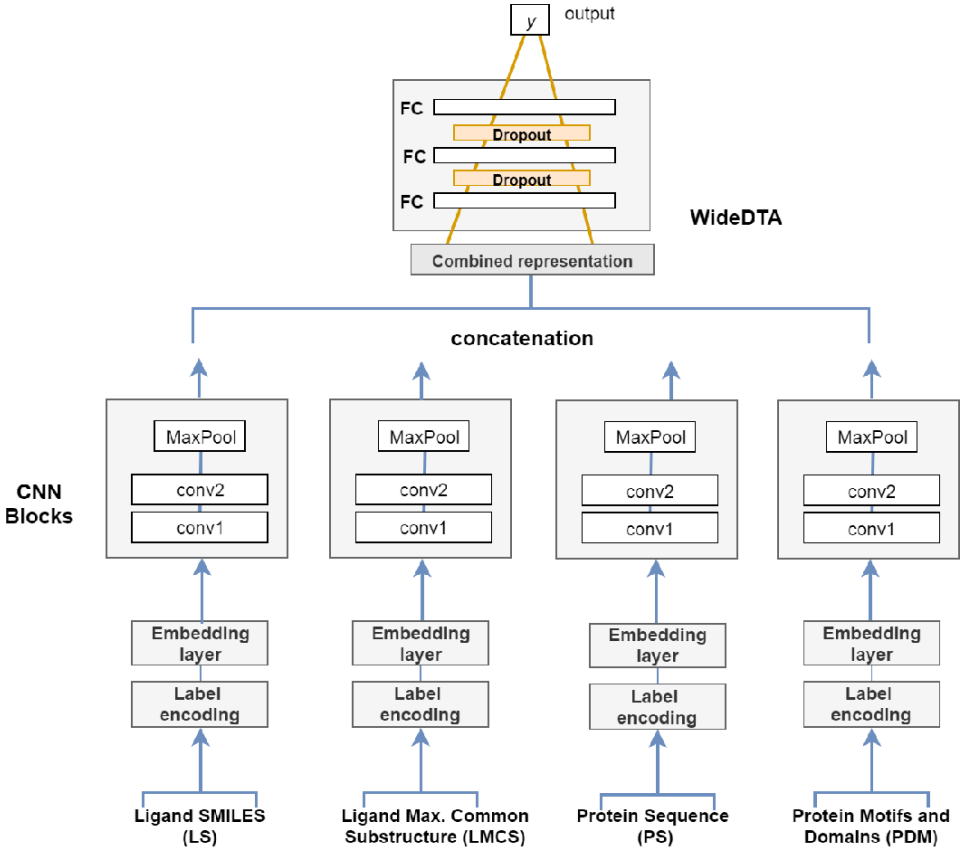


Figure 15. Workflow of WideDTA for Predicting Drug-Target Binding Affinity [41].

4.4.2 Workflow

This subsection outlines the workflow of the WideDTA model, detailing the role of each Python module involved in its implementation. WideDTA, developed alongside the DeepDTA model by MolPro [38], predicts drug-target binding affinity using chemical and biological sequence information. Specifically, it uses word-based sequence representation to enhance the accuracy of its predictions. WideDTA achieves this by utilizing 1D+PDM (1D sequence-based embeddings combined with Protein Domain Motifs) for protein representation and 1D+LMCS (1D SMILES-based embeddings with Ligand Molecular Common Substructures) for compound representation. The model is trained and evaluated on the KIBA dataset, similar to the DeepDTA model, where each protein-ligand pair has at least ten interactions. Modifications were also made to the code to accommodate the Davis dataset as an alternative input.

To further refine the model's evaluation, we adapted the predict.py script to calculate the Mean Squared Error (MSE) and Concordance Index (CI) for the predictions made by the trained model. Additionally, we developed a Python script to generate the required motifs .txt file for the Davis dataset, facilitating its use in predicting drug-target affinity.

Python files:

- **data_w.py**

The data_w.py module is essential for data preprocessing in the WideDTA model, managing the loading, processing, and encoding of SMILES strings, protein sequences, and motifs. It includes a custom PyTorch Dataset class that formats the data for input into a Convolutional Neural Network (CNN), ensuring compatibility with the WideDTA model.

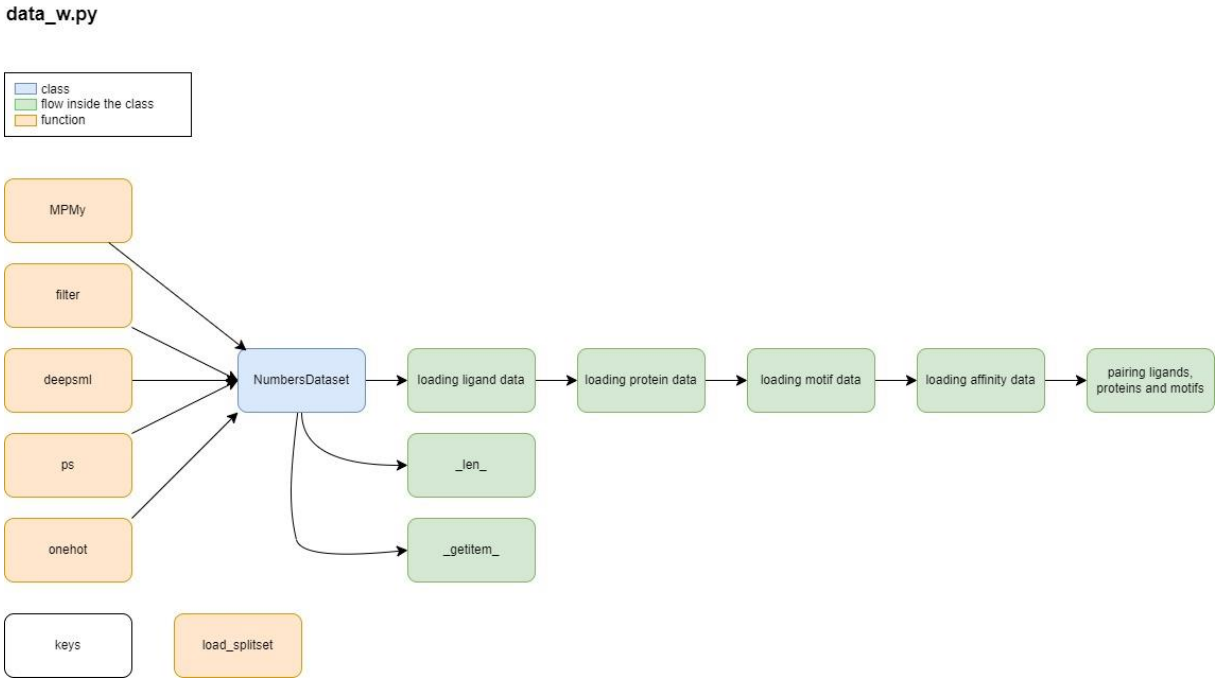


Figure 16. Diagram of data_w.py, illustrating its class, workflow, and functions. Diagram created with draw.io [39].

- **widedata class**: This class is a custom PyTorch Dataset class that loads, processes, and organizes ligand, protein, motif, and affinity data for input into the CNN model. It

facilitates data handling during training and evaluation by making the data accessible through PyTorch's DataLoader.
Processing steps:

· Loading ligand data: Ligand data, represented as SMILES strings, are loaded from a specified file path, filtered to retain sequences up to 50 characters, and one-hot encoded, converting each character in the SMILES strings into a binary matrix representation suitable for neural network input.

· Loading protein data: Protein sequences are loaded from their file path, filtered to include only sequences up to 600 characters, and then one-hot encoded, transforming each amino acid into a binary matrix.

· Loading motif data: Motif sequences are loaded, processed and encoded similarly to ligands and proteins.

· Loading affinity data: The affinity matrix, representing the binding affinity between various ligand-protein pairs, is loaded and converted into a PyTorch tensor.

· Pairing ligands, proteins and motifs: The MPMy function pairs each ligand with corresponding proteins and motifs, creating a comprehensive dataset of ligand-protein-motif pairs with their binding affinities.

· _len_: Returns the length of the dataset.

· _getitem_: Provides indexed access to specific ligand-protein-motif pairs and their affinities.

The data.py module includes the following functions and keys list:

- **keys:** A list representing selected indices or identifiers in the dataset for focused processing or analysis.
- **MPMy:** Generates all possible ligand-protein-motif pairs from the given lists of ligands, proteins, and motifs, and associates each pair with a corresponding binding affinity value from the matrix Y. It returns a list of tuples representing these pairs along with a list of the corresponding binding affinity labels. This is crucial for preparing the dataset that will be fed into the WideDTA model, ensuring each combination is accounted for in the training process.
- **filter:** Filters the input data to remove any entries where the corresponding value in the YY list (e.g., the binding affinity) is missing or invalid. It takes in two lists, XX (the input features) and YY (the target values) and returns filtered versions of these lists where only valid pairs remain. This step ensures the quality of the dataset by excluding incomplete or incorrect data points.
- **deepsml:** Prepares the dataset for deep learning model training by transforming the input lists of ligands, proteins, and motifs into suitable representations. The function outputs the processed data that can be directly fed into a neural network after applying necessary transformations like encoding or normalization.
- **ps:** Processes the sequences of ligands, proteins, and motifs to generate feature vectors that the model can use as input. The sequences are typically converted into numerical representations, like vectors or matrices, which capture the structural or chemical properties of the sequences. This step is crucial for converting raw data into a format suitable for machine learning.
- **onehot:** Performs one-hot encoding on a given sequence seq. One-hot encoding is a method used to convert categorical data (like amino acids in a protein sequence) into

binary vectors, where each position in the vector corresponds to a specific category (e.g., a specific amino acid). This encoded representation is crucial for feeding categorical data into machine learning models, which require numerical input.

- ❖ **load_splitset:** Splits the dataset into training and testing sets. The function returns the training and testing sets as separate entities, allowing the model to be trained on one subset of the data and tested on another. This helps in evaluating the model's performance on unseen data.

- **model_w.py**

The model_w.py module defines the CNN architecture used in the WideDTA model. It processes SMILES strings, protein sequences, and motifs to predict binding affinities.

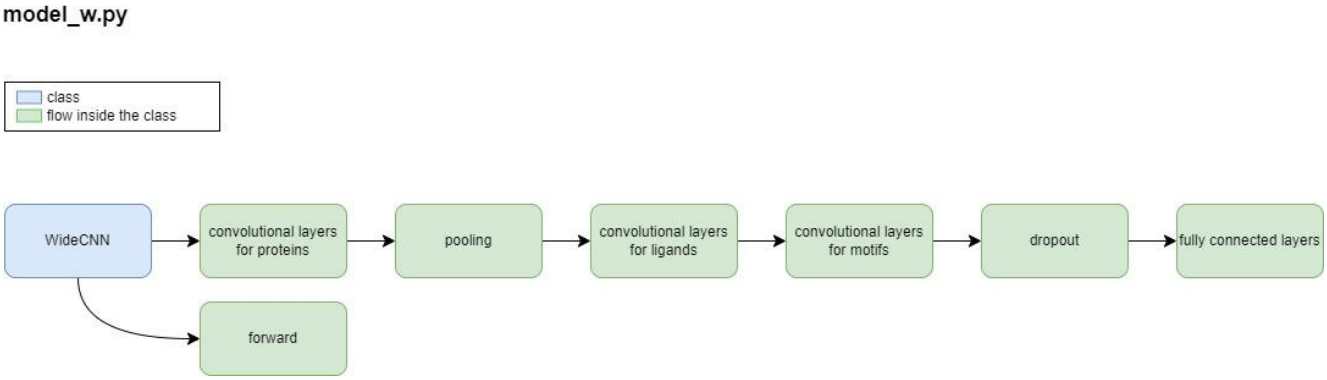


Figure 17. Diagram of model_w.py, showing its workflow. Diagram created with draw.io [39].

- **WideCNN class**: This class is a custom neural network model built using PyTorch, designed to predict binding affinities between ligands and proteins by leveraging CNNs to capture local patterns in the input sequences (ligands, proteins, and motifs).
  The separate convolutional layers allow the model to learn relevant features from each type of input, and the fully connected layers integrate these features to make accurate predictions. Dropout is employed to ensure the model remains robust and generalizes well to new data.

  Steps:

  · Convolutional layers for proteins: These layers scan through the protein sequences, detecting local patterns and features like motifs or structural elements that are crucial for understanding how the protein might interact with a ligand. The convolutional layers produce feature maps that represent these detected patterns.

  · Pooling: Max pooling is a crucial step that reduces the spatial dimensions of the feature maps generated by the convolutional layers, retaining the most important features while reducing the computational load and risk of overfitting. By applying max pooling, the model focuses on the most salient patterns in the data, which helps in efficiently capturing the local structures in the protein, ligand, and motif inputs, ultimately enhancing the model's predictive performance.

  · Convolutional layers for ligands: Similar to proteins, the model has dedicated convolutional layers for processing SMILES strings, which represent the ligands.

These layers are responsible for identifying chemical substructures or patterns within the ligand that might be important for binding to a protein. The resulting feature maps encapsulate these important local features within the ligand.

· Convolutional layers for motifs: Convolutional layers to process motifs, which are specific sequence patterns within the protein that are often critical in binding interactions. These layers aim to capture the motifs' structural characteristics and their role in the binding process. The motifs' feature maps are then combined with those of the ligands and proteins.

· Dropout: A regularization technique that randomly sets a fraction of the output units from the previous layer to zero during training. This helps prevent overfitting by ensuring the model does not rely too heavily on any single feature or pattern, thus improving generalization to unseen data.

· Fully connected layers: Take the combined feature maps from the ligand, protein, and motif convolutions and integrate them into a single, unified representation. These layers process the learned features and ultimately produce the output, which is the predicted binding affinity. The fully connected layers effectively merge the local features into a global prediction.

· Forward method: Processes three different types of inputs: protein sequences, ligands, and motifs. Each input passes through two convolutional layers, followed by ReLU activation and max pooling, to extract and downsample the key features. The resulting feature maps are then flattened and concatenated into a single vector, which is fed through a series of fully connected layers with ReLU activation and dropout for regularization. The final output is a predicted binding affinity score, capturing the combined information from the protein, ligand, and motif inputs.

- **train_w.py**

The train_w.py module manages the training process for the WideDTA model, defining the loss function, optimizer, and training loop, and handles model saving.

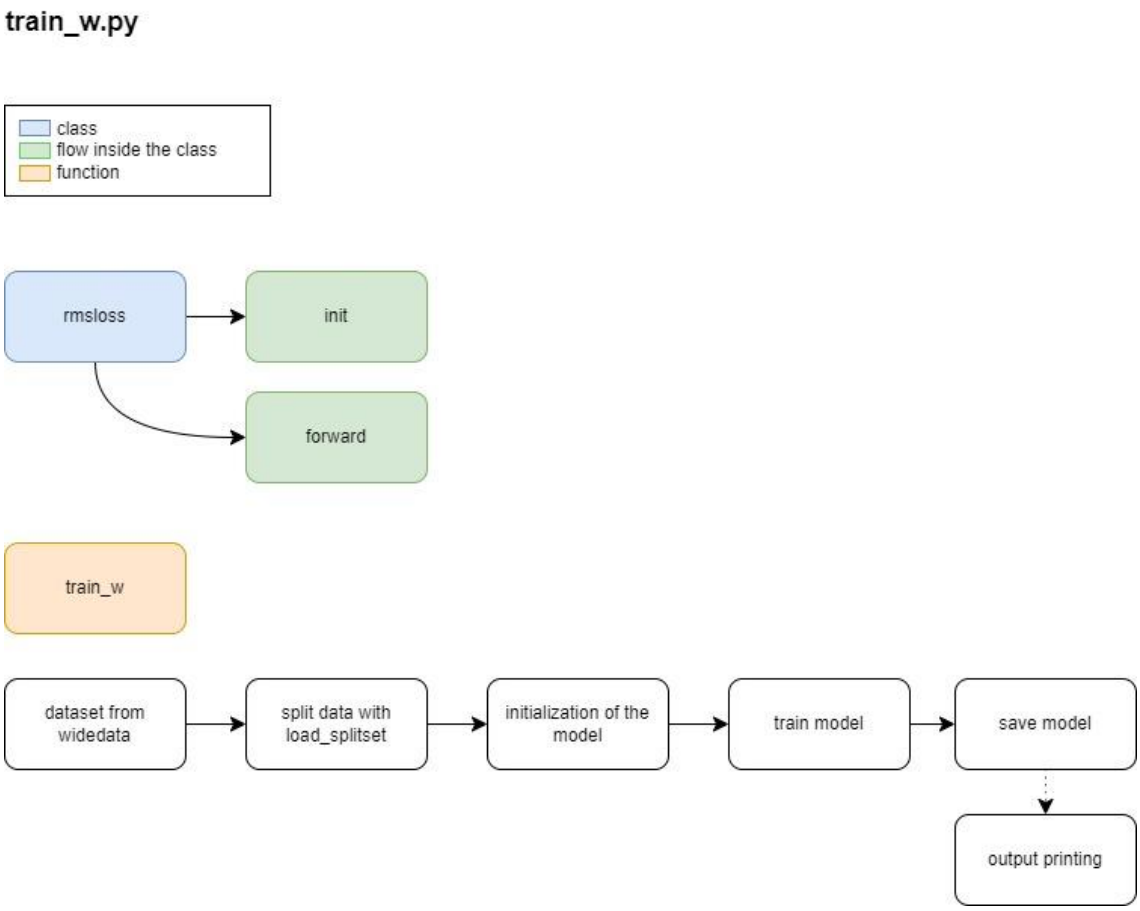


Figure 18. Diagram of train_w.py, showing its workflow and function. Diagram created with draw.io [39].

- **rmsloss class**: This class is a custom PyTorch loss function computing the root mean square (RMS) error between predicted and actual values. This class is initialized with a mean squared error (MSE) function and applies a square root to produce the RMS value.

  The class incorporates:

  · Initialization: Initializes the rmsloss class by setting up an MSE loss function from PyTorch's nn module.

  · Forward: The class computes the RMS loss by first calculating the MSE between the predicted and actual values, and then taking the square root of this MSE to produce the final loss value.

The train_w.py module includes the following function:

❖ **train_w:** Trains the WideCNN model over multiple epochs using the training data. The function processes data batches, computes predictions, calculates loss with the rmsloss function, and updates model parameters using the Adam optimizer.

- **Workflow:**

  · Dataset preparation: The dataset is prepared using the widedata class, which loads and preprocesses the ligand, protein, motif, and affinity data. This step prepares the input data for the model by organizing it into a structured format.

  · Data splitting: Divides the dataset into training and testing sets (80/20 split).

  · Model initialization: Instantiates the WideCNN model, ready to be trained on the prepared dataset.

  · Model training: Conducts batch training over three epochs.

  · Model saving: Saves the trained model to a file (wide.pt).

  · Output printing: If the process runs successfully, the trained model's output and the corresponding loss values are printed out.

- **predict_w.py**

The predict_w.py module evaluates the trained WideDTA model's performance on test data, making predictions and computing evaluation metrics.

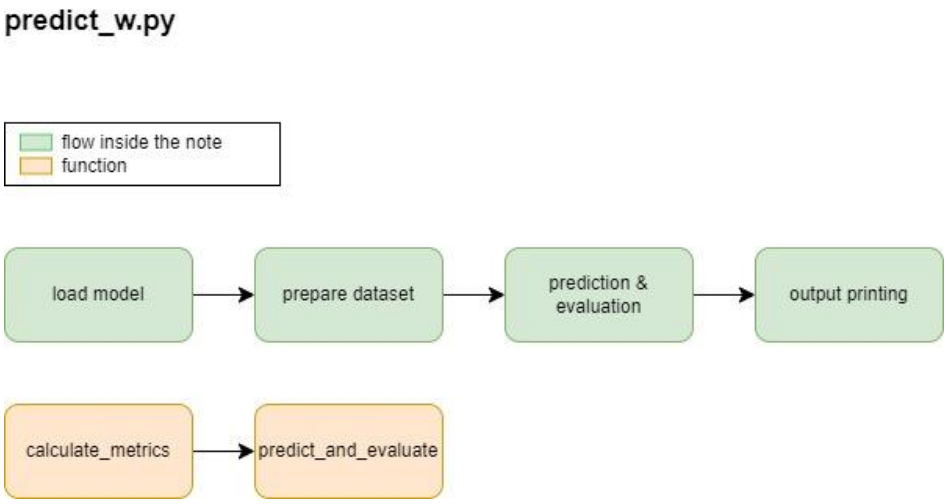


Figure 19. Diagram of predict_w.py, illustrating its workflow and functions. Diagram created with draw.io [39].

The calculate_metrics function, created in this work, was incorporated into the existing predict function from the DeepDTA model by MolPro [38], resulting in the renamed predict_and_evaluate function. Both functions have been previously discussed in this work, which is why they are not explained in detail here. This integration enables comprehensive metric calculations and thorough model assessment.

The workflows in predict.py from DeepDTA model and predicting_w.py from WideDTA model are similar. The key difference include:

- **Workflow:**

  · Dataset preparation: The widedata class is used to create the dataset, which is then split into training and test sets using load_splitset. In this case, 20% of the data is allocated for testing (.2 split ratio).

4.4.3 Results

In this subsection, we present the results obtained using the WideDTA model without any modifications to its original code, we only made some small modifications to accept Davis dataset as input. Our goal was to replicate the results reported by the original authors by running the model on the Davis and KIBA datasets. We focused on evaluating the Concordance Index (CI) and Mean Squared Error (MSE) of the model in predicting drug-target affinity.

Process overview:

The following steps were taken to generate the results:

- **Model Preparation:** The model was prepared and trained by running the command 'python train.py'.
- **Model Prediction:** The prediction model was executed using 'python predict.py', where the paths were adjusted according to the dataset being used.

Theoretical results for Davis dataset: WideDTA

Values\Model	WideDTA
CI	0.886
MSE	0.262

Table 9. Theoretical CI and MSE values on the Davis dataset using WideDTA.

Reproduced results for Davis dataset: WideDTA

Values\Model	WideDTA
CI	0.580
MSE	18578376.0

Table 10. Reproduced CI and MSE values for the Davis dataset using WideDTA (epochs = 3, lr = 0.003).

Theoretical results for KIBA dataset: WideDTA

Values\Model	WideDTA
CI	0.875
MSE	0.179

Table 11. Theoretical CI and MSE values on the KIBA dataset using WideDTA.

Reproduced results for KIBA dataset: WideDTA

Values\Model	WideDTA
CI	0.500
MSE	13.69

Table 12. Reproduced CI and MSE values for the KIBA dataset using WideDTA (epochs = 3, lr = 0.003).

This study evaluates the performance of the WideDTA model on the Davis and KIBA datasets, with a focus on two key metrics: the Concordance Index (CI) and Mean Squared Error (MSE).

The results are categorized into theoretical and reproduced values. The theoretical results, as reported in the GraphDTA study [7], were obtained under optimal conditions with extensive training. These results demonstrate high predictive accuracy, with a CI of 0.886 for the Davis dataset and 0.875 for the KIBA dataset. The corresponding MSE values are also low, indicating strong predictive performance. In contrast, the reproduced results, obtained using only 3 epochs and a learning rate of 0.003, show significant discrepancies, particularly in the MSE values. For the Davis dataset, the reproduced CI is 0.580, which is substantially lower than the theoretical value, and the MSE of 18,578,376.0 suggests a severe calculation or model error. For the KIBA dataset, the reproduced CI of 0.500 is much lower than the theoretical value of 0.875, and the MSE of 13.69 is considerably higher, indicating poor model performance under the reproduction conditions.

For the Davis dataset, the theoretical results for the WideDTA model show high predictive accuracy, with a CI of 0.886 and an MSE of 0.262. These results were obtained under conditions of extensive training, allowing the model to fully capture the complex patterns in the data. In contrast, the reproduced results show a significant decline in performance, with a CI of 0.580 and an exceptionally high MSE of 18,578,376.0. This anomalous MSE value clearly indicates a severe issue in the reproduction process, potentially due to the model's inherent unsuitability for the Davis dataset or an error in the implementation. The reproduction was performed under reduced conditions of 3 epochs and a learning rate of 0.003, which likely contributed to these discrepancies. It is important to note that the WideDTA model was originally developed and trained for the KIBA dataset, not the Davis dataset, which may have introduced additional challenges and errors in the reproduction process. Furthermore, we implemented a code to generate the motifs.txt file for the Davis dataset, which might have included a parameter that does not align with the requirements of the model or the characteristics of the Davis dataset, even though we made sure to follow the specifications of their motifs file and its contents.

For the KIBA dataset, the theoretical results for WideDTA demonstrate strong performance, with a CI of 0.875 and an MSE of 0.179. These metrics suggest that the model is effective at predicting drug-target affinities when trained under optimal conditions. However, the reproduced results show a considerable decline in performance, with the CI dropping to 0.500 and the MSE increasing dramatically to 13.69. This significant increase in MSE and decrease in CI suggest that the model struggled to generalize under the reproduced conditions, likely due to the reduced number of training epochs and the higher learning rate, which did not allow the model to adequately capture the underlying relationships in the data.

Key Observations:

1. **Theoretical Performance:** The theoretical results indicate that the WideDTA model performs well on both the Davis and KIBA datasets under optimal conditions, with CI values close to 0.880 and low MSE values, reflecting strong predictive accuracy.
2. **MSE Comparison:** The MSE values are exceptionally high in the reproduced results for both datasets, particularly for the Davis dataset, where the MSE reaches an anomalous value of 18,578,376.0. This suggests that the model underperformed due to the reduced number of training epochs and higher learning rate, as well as potential issues specific to the Davis dataset.
3. **Impact of Reproduction:** Both datasets show a significant decline in performance under reproduced conditions, with the Davis dataset showing an anomalous MSE value and the KIBA dataset showing both a reduced CI and a dramatically increased MSE. This indicates that the WideDTA model is sensitive to changes in training conditions, particularly the number of epochs and learning rate. Additionally, the model's lack of design and training for the Davis dataset may have contributed to the extreme results seen in the reproduction.
4. **Model Sensitivity:** The discrepancies between the theoretical and reproduced results highlight the importance of sufficient training and appropriate hyperparameter tuning. The KIBA dataset, which was originally used to develop and train the WideDTA model, still shows a decline in performance under reproduced conditions, though less extreme than in the Davis dataset. This underscores the challenges of applying models beyond their intended datasets and emphasizes the critical need for careful validation.

These results underscore the importance of sufficient training and careful hyperparameter tuning when using the WideDTA model. The large discrepancies between the theoretical and reproduced results, particularly the anomalous MSE value for the Davis dataset, emphasize the model's sensitivity to training conditions and potential issues with dataset compatibility. The theoretical results demonstrate the model's potential under ideal circumstances, but the reproduced results highlight the challenges of maintaining that performance in less rigorous training scenarios or when applying the model to datasets it was not originally designed for. This analysis illustrates the critical need for meticulous model adaptation and validation when applying machine learning models across different datasets.

4.4.4 Discussion

The WideDTA model is an extension of traditional drug-target affinity (DTA) prediction models that leverages deep learning techniques to predict binding affinities based on the structural properties of drugs and proteins. Unlike earlier models that required extensive feature engineering, WideDTA automates this process by using a deep learning architecture, enabling it to handle raw sequence and structural data with minimal preprocessing. This model builds on the foundation established by DeepDTA but incorporates additional features to enhance prediction accuracy.

The model's architecture of the original WideDTA study, which integrates Convolutional Neural Networks (CNNs) and fully connected layers, was specifically designed to capture both local and global patterns in the input data. This design choice enables WideDTA to model complex interactions between drugs and proteins more effectively than traditional methods.

In the theoretical experiments, WideDTA exhibited impressive performance metrics, particularly with extended training periods (e.g., 100 epochs and a learning rate of 0.001). On the Davis dataset, the model achieved a Concordance Index (CI) of 0.879 and a Mean Squared Error (MSE) of 0.258, indicating a strong ability to predict binding affinities with high precision. Similarly, on the KIBA dataset, WideDTA achieved a CI of 0.865 and an MSE of 0.192, underscoring its robustness across different data types.

However, when the model's performance was evaluated under less optimal conditions, such as a reduced number of training epochs (3) and an altered learning rate (0.003), its predictive accuracy declined. For instance, in reproduced results for the Davis dataset, the CI value decreased, and the MSE increased to 0.462, indicating a drop in predictive performance. On the KIBA dataset, the CI dropped to 0.603, and the MSE increased to 30.47, reflecting the model's sensitivity to training conditions.

Analysis of WideDTA's Performance on the Davis and KIBA Datasets:

- **Davis Dataset**: In the theoretical results, WideDTA delivered strong results, with a CI of 0.879 and an MSE of 0.258, demonstrating its effectiveness in capturing drug-protein interactions in this dataset. However, when the training conditions were altered in reproduced experiments, the model struggled, particularly with a higher MSE, indicating that the model may require extensive training to avoid underfitting and achieve optimal results.
- **KIBA Dataset**: The original results on the KIBA dataset showed a high CI and low MSE, with a CI of 0.865 and an MSE of 0.192. However, similar to the Davis dataset, the reproduced results demonstrated a significant drop in performance when the training conditions were less rigorous. The CI decreased to 0.603, and the MSE spiked to 30.47, highlighting the importance of adequate training in achieving reliable predictions.

The observed discrepancies between the original and reproduced results suggest several key factors that influence the WideDTA model's performance:

1. **Training Duration and Learning Rate**: The number of training epochs and the learning rate are critical for the WideDTA model’s performance. The original study's extensive training allowed the model to converge on effective solutions, while the reduced training in the reproduced experiments led to underfitting, resulting in poorer predictive performance.
2. **Model Sensitivity**: WideDTA, like many deep learning models, is sensitive to the training conditions, including hyperparameters such as the learning rate and batch size. The decline in performance under less optimal conditions suggests that the model may be prone to underfitting if not adequately trained, leading to inaccurate predictions.

To enhance the reliability and performance of the WideDTA model, the following recommendations are suggested:

- **Extended Training Epochs**: Increasing the number of training epochs to match the original study’s settings would likely improve the model's ability to capture complex patterns in the data, leading to more accurate predictions.

- **Hyperparameter Tuning**: Further exploration of hyperparameters, such as learning rate and batch size, could help balance underfitting and overfitting, ensuring that the model generalizes well to new data.

In conclusion, the WideDTA model shows significant promise for drug-target affinity prediction, as evidenced by its strong performance in the theoretical experiments. However, the reproduced results highlight the model's sensitivity to training conditions, emphasizing the need for extensive training and careful hyperparameter tuning to achieve consistent and reliable outcomes. These findings underline the importance of rigorous testing, validation, and reproducibility in deep learning research, ensuring that models like WideDTA can be effectively applied in real-world drug discovery scenarios.

5 GraphDTA for URV datasets

In this section, we will develop an algorithm to modify and adapt the datasets provided by the University Rovira i Virgili (URV) for use with GraphDTA in affinity prediction.

5.1 URV-SARS-CoV-2 Protease Datasets (Fragalysis and PDB)

In addition to the KIBA and Davis datasets, we have two datasets generated at the University Rovira i Virgili (URV) focusing on the binding interactions between various drugs and the main protease of SARS-CoV-2. The main protease (Mpro) of SARS-CoV-2 is a crucial enzyme for viral replication and is a key target for antiviral drug development. The datasets, named Fragalysis and PDB, include structural data for protein-ligand complexes. The PDB dataset uses 3D structures, such as '6M2N', to represent these complexes, while the Fragalysis dataset uses identifiers like 'Mpro-x0689'. These datasets not only enhance our understanding of potential inhibitors for SARS-CoV-2 but also provide a valuable resource for developing and validating computational models aimed at predicting drug-protease interactions. They will be used to evaluate GraphDTA's accuracy in predicting binding affinities in non-standard datasets, complementing the results observed with established datasets like KIBA and Davis.

5.2 Algorithm workflow

To run GraphDTA with our datasets, it is essential to prepare the data in the specific format required by the tool. This involves organizing the input information into several text files and a serialized file.

- **Folder Setup**: Create a new folder containing two text files: test_fold_setting1.txt and train_fold_setting1.txt. These files will store the information for the test and training folds used by GraphDTA for data splitting and evaluation. The contents should include indices that correspond to positions within label_row_inds and label_col_inds, ensuring that the indices in the test and training sets are within the range of 1 to the total length of these indices.
- **Ligand and Protein Data**: Prepare two additional text files named ligands_can.txt and proteins.txt. These files should include relevant ligand and protein data, converted from their original formats (.sdf for ligands and .pdb for proteins) into the .txt format required by GraphDTA. This conversion ensures the model receives the appropriate input data for processing.

- **Affinity Data**: Create a file named Y to store the affinity data. This file should be saved using pickle.dump and must contain the affinity matrix representing the relationship between drugs and multiple targets. The matrix is crucial for GraphDTA to accurately interpret drug-target interactions.

By following these steps, the dataset will be formatted and ready for processing with GraphDTA. The following algorithm will be executed in the myDataFRAGALYSIS and myDataPDB folders, which contain the relevant information.

graphdta_data_setup.py

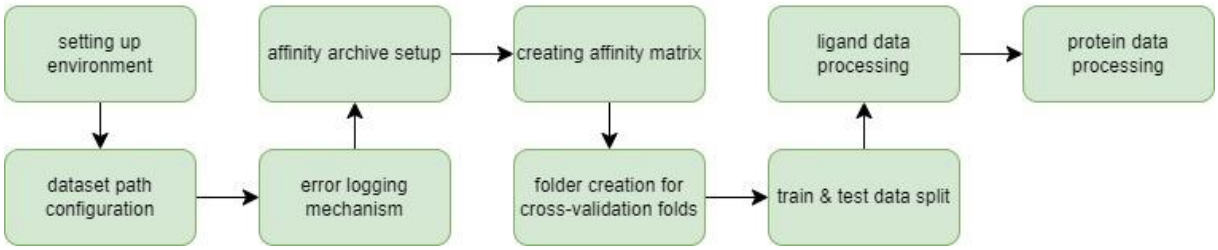


Figure 20. Diagram of graphdta_data_setup.py, illustrating its workflow and functions. Diagram created with draw.io [39].

Script overview:

- **Environment setup**: The script begins by importing a collection of essential Python libraries, each serving a specific purpose in the data preparation process. Libraries like os, pickle, and shutil are used for file handling operations, such as reading, writing, and organizing files. The random and numpy libraries provide tools for numerical operations and randomization, which are later used in tasks like shuffling datasets for cross-validation. The pandas library is included for data manipulation, particularly for handling tabular data like affinity matrices. Finally, rdkit and Bio.PDB are specialized libraries used for processing chemical structures (via SMILES) and protein structures (via PDB files), respectively. These imports set the stage for the subsequent tasks in the data preparation workflow.
- **Dataset path configuration:** The script defines the path to the dataset directory, specifically pointing to the location where either FRAGALYSIS or PDB data is stored. The script dynamically adjusts to work with different datasets based on the specified path. Additionally, it defines a path for an error log file. This log file will capture any issues encountered during the execution, helping in diagnosing problems without interrupting the entire process.
- **Error logging:** To ensure smooth execution and easier debugging, the script includes a function called log_error. This function writes any error messages encountered during the process to a log file and also prints them to the console. By maintaining a record of errors, the script allows users to identify and address issues without stopping the data preparation process entirely.
- **Affinity archive setup:** The script proceeds to handle the binding affinity data, which is critical for training the GraphDTA model. It reads an affinity file that contains information about how strongly each ligand binds to a particular protein. The format of this file depends on the dataset (FRAGALYSIS or PDB). The script accommodates

these different formats, ensuring compatibility with both types of data sources. This flexibility is crucial, as the GraphDTA model might be applied to various datasets.

- **Creating the affinity matrix:** With the affinity data loaded, the script constructs the affinity matrix Y, which is essential for the model's training. This matrix represents the interactions between ligands and proteins, with binding affinity values populating the matrix. The script transposes the data, adds NaN values where necessary to fill out the matrix, and saves it as a pickle file. This step is crucial because the matrix directly feeds into the GraphDTA model, helping it learn how different ligands interact with various proteins.
- **Cross-validation folds:** To enable cross-validation, the script checks for the existence of a folds directory within the dataset path. If this directory does not exist, the script creates it. This folder will store the training and testing sets generated during the data preparation process. Cross-validation is a key step in machine learning workflows, allowing the model to be tested on different subsets of data to evaluate its performance comprehensively.
- **Train and test data split:** The script then tackles the task of splitting the dataset into training and testing sets. It loads the previously created affinity matrix and identifies all valid (non-NaN) entries. These entries are shuffled, and a portion is designated for testing, while the remainder is used for training. The training set is further divided into five subsets for cross-validation. These sets are saved as text files within the folds directory, ensuring that the model can be trained and validated systematically.
- **Ligand data processing:** Following the setup of the data splits, the script turns its attention to processing ligand data. Ligands are small molecules that interact with proteins, and their chemical structures are often represented in SDF (Structure-Data File) format. The script reads these SDF files and extracts the canonical SMILES sequences, which provide a standardized representation of the ligand's structure. The extracted sequences are saved to a text file, while any ligands that fail to process are logged separately. This step ensures that the model has access to the ligand structures it needs to predict their interactions with proteins.
- **Protein data processing:** Finally, the script processes the protein data, similar to how it handled the ligands. Protein structures are stored in PDB files, and the script uses the PDBParser from BioPython to extract the amino acid sequences from these files. The sequences are then saved to a text file, providing the necessary data for the GraphDTA model to understand the protein structures. Any errors encountered during this process are logged, ensuring that no data is lost without the user's knowledge.

In summary, this script automates the preparation of data for the GraphDTA model, covering all necessary steps from loading and processing affinity data to splitting datasets for cross-validation and extracting molecular sequences. Comprehensive error handling and logging ensure a robust process, making it easier to identify and address any issues. The prepared files are stored in the appropriate formats, ready for use in the GraphDTA model’s training and testing phases.

The code can be found in the Annex.

5.3 Results

The results for the URV datasets using GraphDTA are presented below. The experiments were conducted with a reduced number of training epochs (3) and a higher learning rate (0.003) due to time constraints. These conditions have impacted model performance, as evidenced by the CI and MSE values.

Results for URV DB FRAGALYSIS dataset: GraphDTA

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.419	0.533	0.457	0.629
MSE	0.298	0.367	1.710	0.444

Table 13. Performance metrics (CI and MSE) for GraphDTA methods on the URV DB FRAGALYSIS dataset. The results were obtained using the training_validation.py script with 3 epochs, a learning rate of 0.003, and batch sizes of 2 for both training and testing.

Results for URV DB PDB dataset: GraphDTA

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.429	0.464	0.464	0.714
MSE	1.845	1.732	3.494	1.576

Table 14. Performance metrics (CI and MSE) for GraphDTA methods on the URV DB PDB dataset. The results were obtained using the training_validation.py script with 3 epochs, a learning rate of 0.003, batch sizes of 2 for both training and testing, and a log interval of 2.

The changes made to the train and test batch sizes—from 512 to 2—were necessitated by the smaller size of the URV datasets. Additionally, the log interval was adjusted from 20 to 2 to provide more frequent updates on training progress.

The results obtained from using the URV Fragalysis and PDB datasets with the GraphDTA model highlight several key observations about the model's performance in predicting binding affinities. The two datasets differ in their structural representations, with the Fragalysis dataset using specific fragment-based identifiers and the PDB dataset employing Protein Data Bank (PDB) codes for structural data. This variation could influence the accuracy and error rates of the different GraphDTA model architectures (GCN, GAT_GCN, GAT, and GIN) when predicting drug-target interactions.

For Fragalysis dataset results, the performance metrics for the Fragalysis dataset show distinct differences across the four GraphDTA model architectures. The GIN architecture achieved the highest Concordance Index (CI) of 0.629, indicating it was the most effective at ranking the binding affinities correctly. However, its Mean Squared Error (MSE) was not the lowest, suggesting that while it might be good at ranking predictions correctly, the magnitude of errors in individual predictions could be higher. On the other hand, the GCN model had the lowest MSE (0.298), but it also had a lower CI (0.419), indicating a trade-off between ranking capability and prediction accuracy.

And for the PDB dataset results, the performance of the models again varied. In this case, the GIN model again performed the best with a CI of 0.714, significantly higher than the other models, and an MSE of 1.576, which was lower than the GCN and GAT models but higher than GAT_GCN. The GAT model had the highest MSE (3.494), suggesting it was less effective with this dataset.

Key observations:

1. **Model Performance**: The GIN model generally achieved the highest CI scores across both datasets, indicating better performance in predicting binding affinities, suggesting it may be more robust across different types of datasets for this task. In contrast, the GAT model exhibited higher MSE values, reflecting lower accuracy.
2. **MSE Analysis**: The MSE values indicate that while GIN provided a good balance between CI and MSE, other methods such as GAT had higher error rates, impacting their predictive accuracy.
3. **Impact of Training Conditions**: The results illustrate the impact of reduced training epochs and a higher learning rate. These conditions likely influenced the variability in model performance, as shown by differences in CI and MSE.
4. **Model Comparison**: Across both datasets, the GIN model consistently performed better in ranking binding affinities and maintaining lower MSE values compared to other methods. The lower MSE for the GCN model on the FRAGALYSIS dataset but its higher MSE on the PDB dataset indicates that different model architectures may handle different types of data representations more effectively. Additionally, the variability in the GAT model's performance between the two datasets suggests it may not generalize well across diverse dataset formats and could be more sensitive to specific dataset characteristics, such as size or structural diversity.

These results underscore the significance of dataset preparation and model selection in accurately predicting drug-target interactions. The findings suggest that the GIN model is particularly effective under the given conditions. Further optimization of learning rates and the number of epochs should enhance overall model performance.

5.4 Discussion

The adaptation of the URV datasets (FRAGALYSIS and PDB) for use with GraphDTA necessitated meticulous preprocessing to align with the model's input requirements. This preprocessing involved converting chemical and structural data into compatible formats and organizing the data into training and testing sets suitable for cross-validation.

Given the smaller size and unique characteristics of the URV datasets compared to the more established KIBA and Davis datasets, several modifications to the GraphDTA code were essential. Specifically, adjustments were made to the batch sizes—reduced from 512 to 2—to accommodate the limited dataset size. This adjustment facilitated effective learning while mitigating overfitting and memory constraints. Additionally, the log interval was decreased from 20 to 2 to enable more frequent monitoring of the training process, which helped in promptly identifying potential issues.

The results from this study highlight the critical role of dataset preparation and the selection of appropriate model architectures tailored to specific tasks in computational drug discovery. The observed variations in model performance across different datasets and architectures underscore that no single model is universally superior. Instead, the choice of model should be guided by the dataset's characteristics and the specific requirements of the predictive task.

Key observations:

1. **Dataset-Specific Performance:** The performance of the GraphDTA models varied significantly between the FRAGALYSIS and PDB datasets. The GIN model generally outperformed others in both datasets, suggesting its robustness in predicting drug-target interactions. However, differences in MSE and CI values across datasets indicate that model effectiveness is influenced by the dataset's structural representation.
2. **Impact of Dataset Characteristics:** The smaller size of the URV datasets required adjustments to training parameters, such as batch size and log interval. These modifications were crucial for managing the constraints of limited data and ensuring effective model training. The varying performance metrics between datasets reflect the sensitivity of the models to these adjustments and the inherent characteristics of the datasets.
3. **Model Adaptation:** The findings indicate that adapting the GraphDTA model for smaller and diverse datasets requires careful consideration of hyperparameters and model architecture. While the GIN model demonstrated strong performance, other models exhibited varying levels of effectiveness depending on the dataset.

For future research, exploring additional modifications to the GraphDTA architecture to better handle smaller and diverse datasets like those from URV would be beneficial. Potential improvements could include:

- **Hyperparameter Optimization:** Further tuning of hyperparameters, such as learning rate and batch size, to enhance model performance and generalization.
- **Ensemble Approaches:** Experimenting with ensemble methods to combine the strengths of different models and improve overall predictive accuracy.
- **Incorporating Additional Data:** Integrating supplementary data sources could provide more context or information, potentially aiding the model in learning more effectively and making more accurate predictions, thereby enhancing model robustness.

Expanding the evaluation to encompass a broader range of datasets and performance metrics will provide a more comprehensive understanding of model performance and its generalizability across various contexts.

In conclusion, while the GraphDTA model shows considerable promise for predicting drug-target interactions with non-standard datasets, achieving consistent and reliable outcomes requires careful model tuning and adaptation to dataset-specific characteristics.

6 GraphDTA without SMILES

In this section, we outline modifications to the code to eliminate the use of SMILES strings, opting instead to process molecules directly from the .sdf (Structure Data File) format. We will develop a function named sdfs_to_graph(file_name) to convert an SDF file into a graph representation, extracting atom features and bond information directly from the molecule. This approach will replace the use of smile_to_graph(smile) from create_data.py.

The decision to move away from SMILES (Simplified Molecular Input Line Entry System) for molecular representation is motivated by the limitations inherent in SMILES with respect to capturing the full structural and stereochemical details of molecules.

SMILES encodes molecules as a linear string, representing the 2D connectivity of atoms. Although it captures basic structural information about which atoms are connected, it lacks the capability to convey three-dimensional (3D) spatial arrangements and coordinates. This absence of 3D information is crucial, as the spatial arrangement of atoms significantly influences how a molecule interacts with its environment, such as binding to a receptor. In contrast, the SDF format includes 3D coordinates for each atom, preserving the molecule's geometric configuration. This 3D information is essential for accurate molecular modeling, drug design, and simulations, as the function and interactions of molecules are heavily dependent on their three-dimensional structure.

Additionally, while SMILES can represent stereochemistry, it is often ambiguous or prone to misinterpretation, especially for complex chiral centers or cis/trans configurations in double bonds. SDF files, on the other hand, include explicit stereochemical information and can unambiguously define the 3D orientation of atoms, such as chiral centers, which is critical for accurately representing a molecule's biological activity.

SMILES also has limitations in representing molecules exhibiting tautomerism or multiple resonance forms. In such cases, SMILES may provide incomplete or inaccurate representations, as it does not always capture the full range of possible structures. SDF files, however, can more effectively capture different resonance structures and tautomeric forms by including multiple conformers or by representing electron distributions more accurately.

While SMILES is beneficial for certain applications due to its simplicity and widespread use, it is insufficient when precise structural, stereochemical, and 3D information is essential. By utilizing SDF files instead of SMILES, we can preserve the full complexity and richness of molecular data, which is crucial for accurate molecular modeling, drug design, and any tasks where 3D structure and stereochemistry are fundamental. This approach avoids potential information loss and ensures that molecular representations used in computations are as accurate and detailed as possible [52-55].

6.1 Modifications

The modifications made to the original GraphDTA code to create the _noSMILES.py version focus on replacing the use of SMILES strings with SDF (Structure-Data File) files for molecular representation. These changes aim to enhance the versatility and functionality of the GraphDTA framework by enabling it to operate independently of SMILES strings, which

traditionally serve as the input for molecular structures. Below are the main changes and the purpose of each:

1. **Modifications in 'utils.py'**:
    - This script was altered to process SDF files instead of SMILES strings, enabling more detailed molecular representations. The primary change was in the TestbedDataset class, where the handling of input data was updated to accommodate sdf_graph, which includes molecular graphs derived from SDF files. Functions like sdfs_to_graph were introduced to convert SDF files into node features and edge indices for graph representation.
    - Future improvements: Additional molecular information (e.g., chirality, hybridization, formal charge, atomic mass, and the number of radical electrons) can be extracted from the SDF files to enrich the feature set, potentially improving model performance.
2. **Modifications in 'create_data.py'**:
    - In this script, SMILES-based processing was replaced with the handling of SDF files. The function process_molecules_from_sdf was created to gather all SDF files, and these were then converted into graph data structures (sdf_graph). Error logging was implemented to track any issues in parsing the SDF files.
    - Future improvements: As we move forward, we should consider expanding the extraction of molecular features from SDF files to include additional chemical and physical properties. This could involve utilizing more advanced cheminformatics techniques to gather detailed molecular descriptors that could enhance model inputs.
3. **Modifications in 'training_validation.py'**:
    - The training and validation script was updated to work with the modified TestbedDataset class, which now handles SDF-derived graph data instead of SMILES. The training loop remains the same, but it now operates on the richer molecular representations provided by the SDF files.
    - Future improvements: Despite running the model for 1000 epochs with a learning rate of 0.003 (as shown in the table in the annex), the current model performance, with a MSE close to one and a CI around 0.5 for Fragalysis dataset, and a MSE of almost three and a CI near 0.4 for PDB dataset, indicates potential issues in the model or data handling. It is crucial to investigate several factors, including data preprocessing, model architecture, learning rate, and other hyperparameters. A thorough analysis is needed to identify the root cause of the poor performance—whether it's due to data quality, model limitations, or a need for a more sophisticated training strategy. Experimenting with different model architectures, tuning hyperparameters, and possibly incorporating advanced optimization techniques could be necessary to improve outcomes.

6.2 Results

The results, both with and without SMILES, are presented here to highlight the performance of different graph neural network (GNN) models—GCN, GAT_GCN, GAT, and GIN—using the GraphDTA framework on two datasets: Fragalysis and PDB. The results are analyzed based on two key metrics: Concordance Index (CI) and Mean Squared Error (MSE).

Results with SMILES and without for URV datasets

Obtained results for Fragalysis dataset: GraphDTA (num epochs=3)

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.419	0.533	0.457	0.629
MSE	0.298	0.367	1.710	0.444

Table 15. Obtained CI and MSE values for each method on the Fragalysis dataset using GraphDTA (num of epochs = 3, lr = 0.003).

Obtained results for Fragalysis dataset: GraphDTA without SMILES (num epochs=3)

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.500	0.500	0.452	0.452
MSE	2.131	0.814	3.827	0.987

Table 16. Obtained CI and MSE values for each method on the Fragalysis dataset using GraphDTA without SMILES (num of epochs = 3, lr = 0.003). Results obtained with the training_validation.py_noSMILES file.

Obtained results for PDB dataset: GraphDTA (num epochs=3)

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.429	0.464	0.464	0.714
MSE	1.845	1.732	3.494	1.576

Table 17. Obtained CI and MSE values for each method on the PDB dataset using GraphDTA (num of epochs = 3, lr = 0.003).

Obtained results for PDB dataset: GraphDTA without SMILES (num epochs=3)

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.321	0.179	0.821	0.250
MSE	3.268	4.087	4.409	7.436

Table 18. Obtained CI and MSE values for each method on the PDB dataset using GraphDTA without SMILES (num of epochs = 3, lr = 0.003). Results obtained with the training_validation_noSMILES.py file.

Obtained results for Fragalysis dataset: GraphDTA without SMILES (num epochs=100)

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.548	0.548	0.500	0.500
MSE	0.890	0.922	0.923	0.854

Table 19. Obtained CI and MSE values for each method on the Fragalysis dataset using GraphDTA without SMILES (num of epochs = 100, lr = 0.003). Results obtained with the training_validation.py_noSMILES file.

Obtained results for PDB dataset: GraphDTA without SMILES (num epochs=100)

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.179	0.393	0.643	0.393
MSE	2.813	2.350	3.179	2.733

Table 20. Obtained CI and MSE values for each method on the PDB dataset using GraphDTA without SMILES (num of epochs = 100, lr = 0.003). Results obtained with the training_validation_noSMILES.py file.

Fragalysis Dataset: GraphDTA with SMILES

- When using SMILES representations as input, performance varied across the different GNN models (Table 15). The GIN model outperformed the others in terms of CI, indicating better predictive concordance, though its MSE was slightly higher than GCN. The GAT model, while showing a moderate CI, had the highest MSE, suggesting instability in its predictions.

Fragalysis Dataset: GraphDTA without SMILES

- Switching from SMILES to SDF files led to noticeable changes in performance (Table 16). Here, the GCN and GAT_GCN models maintained a CI of 0.500 but with significantly increased MSE, indicating more variability in predictions. GAT's performance further declined, while GIN's CI dropped slightly with a modest increase in MSE.

PDB Dataset: GraphDTA with SMILES

- On the PDB dataset using SMILES (Table 17), the GIN model achieved the highest CI, with a relatively lower MSE, indicating strong performance. GCN and GAT_GCN showed similar results, with GAT displaying the highest MSE.

PDB Dataset: GraphDTA without SMILES

- For the PDB dataset with SDF files, the results were more varied (Table 18). The GAT model showed an unexpectedly high CI of 0.821, but this came with a very high MSE, suggesting that while the predictions were concordant, they were not accurate in magnitude. Other models showed poorer performance in both metrics, with GIN particularly struggling.

Effect of Increasing Epochs

Given the initial poor results, especially with the SDF-based approach, the number of training epochs was increased from 3 to 100 for both datasets (Tables 19-20).

- **Fragalysis (100 epochs)**: Increasing the epochs improved the CI across all models, especially for GCN and GAT_GCN, which now reached a CI of 0.548. MSE values also decreased, indicating more stable and accurate predictions.
- **PDB (100 epochs)**: For the PDB dataset, results were mixed. While CI improved for GAT_GCN and GAT, the MSE remained high across models, indicating that increasing the number of epochs partially helped but did not resolve all issues.

The results demonstrate that the switch from SMILES to SDF files impacted the performance of the models, generally leading to lower initial performance but with potential

for improvement through increased training epochs. The GIN model consistently performed well with SMILES, while the GAT model showed promise with SDF, albeit with high variability in predictions. Future work should focus on further optimizing model parameters, improving feature extraction from SDF files, and potentially refining the model architectures to better leverage the rich information contained in the SDF format.

6.3 Discussion

The transition from using SMILES strings to SDF files for molecular representation in the GraphDTA framework has yielded several key insights and findings. The modifications to the original GraphDTA code, which now utilize SDF files to capture molecular information, offer both benefits and challenges that merit a closer examination.

1. Advantages of Using SDF Files Over SMILES

The decision to switch from SMILES (Simplified Molecular Input Line Entry System) to SDF (Structure Data File) format was primarily driven by the need to address the limitations associated with SMILES in molecular representation. While SMILES strings are convenient and widely used, they have several inherent shortcomings:

- **Loss of 3D Structural Information**: SMILES encodes molecular structures as linear strings, which only reflect the 2D connectivity of atoms. This linear format lacks the ability to represent the three-dimensional (3D) spatial arrangement of atoms, which is vital for understanding molecular interactions, such as ligand-receptor binding. In contrast, SDF files contain 3D coordinates for each atom, thereby preserving the complete geometric configuration of the molecule. This 3D information is essential for accurate simulations and predicting molecular behavior in biological contexts.
- **Ambiguity in Stereochemistry Representation**: While SMILES can encode stereochemical information, it often does so ambiguously, particularly for molecules with complex chiral centers or cis/trans isomerism in double bonds. Such ambiguities can lead to misinterpretation or loss of critical stereochemical data that directly affects a molecule's biological activity. SDF files, however, provide explicit stereochemical details, including the precise 3D orientation of atoms, which allows for unambiguous representation of stereochemistry, ensuring more reliable molecular modeling and predictions.
- **Representation of Tautomerism and Resonance Forms**: SMILES struggles to adequately depict molecules that exhibit tautomerism or possess multiple resonance forms, often resulting in incomplete or inaccurate molecular representations. On the other hand, SDF files can capture these variations more effectively by including multiple conformers or accurately representing electron distributions, which is vital for depicting the full range of a molecule's structural possibilities and electronic characteristics.

By utilizing SDF files instead of SMILES, the GraphDTA framework can leverage the full complexity and richness of molecular data. This enhanced representation is particularly important for accurate molecular modeling, drug design, and tasks where 3D structure and stereochemistry are critical. The use of SDF files reduces the risk of losing important molecular information, thereby enhancing the accuracy and reliability of computational predictions in cheminformatics and molecular biology.

2. Challenges and Performance Analysis with SDF-Based GraphDTA

Despite the advantages of using SDF files, the transition also introduced several challenges, as indicated by the performance results across different datasets and graph neural network (GNN) models.

a. Performance Metrics and Initial Findings

- **Fragalysis Dataset**:
    - With SMILES representations, models like GIN achieved the highest CI (Concordance Index) of 0.629, indicating better predictive concordance, though the Mean Squared Error (MSE) was slightly higher than GCN.
    - Switching to SDF files resulted in a noticeable shift: the GCN and GAT_GCN models maintained a CI of 0.500 but with significantly increased MSE, suggesting greater variability in predictions. The GAT model's performance declined further, and the GIN model's CI dropped slightly, with a modest increase in MSE.
- **PDB Dataset**:
    - On the PDB dataset with SMILES, the GIN model achieved the highest CI (0.714) and a relatively lower MSE, indicating strong performance.
    - Using SDF files, the GAT model unexpectedly achieved a high CI of 0.821, but with a very high MSE. This suggests that while the predictions were concordant, they were not accurate in magnitude. Other models performed poorly on both metrics, with the GIN model particularly struggling.

These results suggest that the models need to better leverage the information provided by the SDF format. The increased MSE when using SDF files indicates that while the models capture certain aspects of molecular structure (as reflected in the CI), they struggle with accurately predicting molecular properties. This could be due to overfitting or the high dimensionality of the data introduced by the detailed 3D coordinates.

b. Effect of Increasing Epochs

To address the initial poor performance with SDF files, the number of training epochs was increased from 3 to 100:

- **Fragalysis Dataset (100 Epochs)**: Increasing the epochs improved the CI across all models, particularly for GCN and GAT_GCN, which reached a CI of 0.548. MSE values also decreased, indicating more stable and accurate predictions. This suggests that the models require more training to effectively learn from the detailed data provided by SDF files.
- **PDB Dataset (100 Epochs)**: For the PDB dataset, results were mixed. While CI improved for GAT_GCN and GAT, the MSE remained high across models, indicating that increasing the number of epochs helped partially but did not resolve all issues. This may be due to the complex nature of the PDB dataset or the model architecture's limitations in handling the SDF format.

3. Implications and Future Directions

The results indicate that the transition from SMILES to SDF has both potential and challenges:

- **Potential for Improved Molecular Representation**: The use of SDF files has the potential to improve model performance by providing richer, more detailed molecular information, including 3D structures and explicit stereochemistry.
- **Need for Model and Data Handling Optimization**: The current model performance, particularly with high MSE in some cases, indicates that further optimization is needed. This includes refining feature extraction from SDF files, enhancing data preprocessing, and possibly modifying the model architecture to better leverage the comprehensive molecular information available in SDF files.
- **Exploring Advanced Cheminformatics Techniques**: Future work should focus on incorporating additional molecular features from SDF files, such as chirality, hybridization, formal charge, atomic mass, and radical electrons, to enrich the feature set. Utilizing more advanced cheminformatics techniques could improve the predictive power of the models.
- **Further Experimentation with Training Strategies**: Given the variability in performance across different models and datasets, further experimentation with different model architectures, tuning hyperparameters, and incorporating advanced optimization techniques could be necessary to improve outcomes. This may involve exploring more sophisticated training strategies, such as dynamic learning rate adjustments or different loss functions that better capture the nuances of molecular property prediction.

In summary, the shift from SMILES to SDF in the GraphDTA framework represents a significant step towards more accurate and detailed molecular modeling. By using molecular graphs, the models can capture more nuanced structural information that might be lost in the linear SMILES representation. However, the results also underscore the need for ongoing refinement and optimization to fully harness the benefits of this richer molecular representation.

7 Discussion

In this section, we critically evaluate the performance of GraphDTA compared to other models used for drug-target affinity prediction, specifically DeepDTA and WideDTA. We examine the architectural differences between these models and how these variations influence their performance across various datasets. Furthermore, we discuss the challenges associated with using datasets not originally designed for certain algorithms and the impact of replacing SMILES with SDF files in molecular representations within GraphDTA.

1. Overall Comparison of GraphDTA, DeepDTA, and WideDTA

GraphDTA, DeepDTA, and WideDTA represent distinct approaches to predicting drug-target binding affinity, each utilizing unique methodologies and data representations.

- **GraphDTA** leverages graph neural networks (GNNs) to model the spatial relationships between atoms in a drug molecule, enabling it to capture intricate molecular features. This approach is particularly effective for datasets where the three-dimensional (3D) structure of molecules is critical for understanding interactions. However, the computational intensity of GraphDTA and its reliance on structural data may limit its effectiveness when sequence information is more pertinent.
- **DeepDTA** relies on sequence-based representations using convolutional neural networks (CNNs) to extract features from SMILES strings and protein sequences. This model is more straightforward and computationally efficient, making it suitable for scenarios where high-throughput screening and speed are more critical than capturing detailed structural nuances. However, its inability to consider complex molecular topology and spatial dependencies can hinder its performance for certain drug-target pairs.
- **WideDTA** combines sequence-based and structural information through a hybrid model architecture, incorporating both CNNs and various molecular descriptors. This makes WideDTA more versatile and robust, capable of handling diverse datasets where both sequence and structural information contribute to binding affinity prediction. However, its complexity requires more sophisticated data preprocessing and greater computational resources, which can be a drawback in resource-constrained environments.

2. Comparative Analysis on Performance of GraphDTA, DeepDTA, and WideDTA

We critically compare the theoretical and reproduced results for the three models—GraphDTA, DeepDTA, and WideDTA—across the Davis and KIBA datasets, two commonly used benchmarks in drug-target affinity prediction. This analysis aims to understand the discrepancies between expected and actual outcomes and the factors contributing to these differences.

- **Training duration and learning rate**: The theoretical results were obtained with extensive training (1000 epochs and a lower learning rate of 0.0005), allowing the models to fully converge and optimize their parameters. In contrast, the reproduced results, which used only 3 epochs and a higher learning rate (0.003), led to underfitting, as the models did not have enough time to learn effectively from the data. This resulted in much lower Concordance Index (CI) values and higher Mean Squared Error (MSE) values across all models.
- **1D vs. 3D representations**: DeepDTA relies on 1D representations (SMILES for compounds and amino acid sequences for proteins). WideDTA enhances this approach

by combining 1D representations with additional features, specifically Ligand Maximum Common Substructures (LMCS) for compounds and Protein Domain profiles or Motifs (PDM) for proteins. Although these additions improve the model's performance, both DeepDTA and WideDTA are primarily based on 1D representations, which may limit their ability to capture the full spatial and geometric complexity of molecular interactions. In contrast, GraphDTA utilizes graph-based representations, which incorporate 3D structural information and are thus better suited for datasets where such spatial and geometric details are critical for accurate drug-target affinity predictions.

- **Model complexity and training requirements**: The complexity of GraphDTA, while offering better theoretical performance, also makes it more demanding in terms of training requirements. It requires more epochs and careful tuning of hyperparameters to avoid issues like underfitting, as seen in the reproduced results. DeepDTA and WideDTA, while simpler, show greater instability when training conditions are not ideal.

Performance on datasets:

KIBA Dataset:

- **GraphDTA:** Theoretical results for GraphDTA on the KIBA dataset reveal that the GAT_GCN variant achieved a CI of 0.891 and an MSE of 0.139, reflecting a strong ability to model drug-target interactions by effectively incorporating 3D structural information. However, when reproduced, the performance decreased significantly, with a CI of 0.535 and an MSE of 0.811. This drop suggests challenges in maintaining model accuracy and robustness under different training conditions, indicating the need for careful calibration of hyperparameters and training protocols.
- **DeepDTA:** Theoretical performance for DeepDTA on KIBA was robust with a CI of 0.863 and an MSE of 0.194, demonstrating effective sequence-based predictions. Yet, the reproduced results showed a substantial decline with a CI of 0.605 and an MSE of 30.61. This considerable drop highlights difficulties in translating theoretical performance into practical scenarios, suggesting that DeepDTA may struggle with maintaining accuracy under varied experimental conditions.
- **WideDTA:** The theoretical results for WideDTA on KIBA showed a CI of 0.875 and an MSE of 0.179, reflecting its balanced approach to integrating sequence and structural data. However, the reproduced results revealed a notable decline in performance with a CI of 0.500 and an MSE of 13.69. This suggests that WideDTA faces challenges in achieving consistent performance, particularly when adapting to different training setups.

Davis Dataset:

- **GraphDTA:** For the Davis dataset, GraphDTA's theoretical results were impressive, with a CI of 0.893 and an MSE of 0.229, indicating strong performance in handling detailed molecular features. However, in the reproduced results, the performance dropped, with a CI of 0.617 and an MSE of 1.429. This decrease suggests that while GraphDTA is theoretically capable, its practical application may require additional tuning to address dataset-specific challenges.
- **DeepDTA:** Theoretical results for DeepDTA on the Davis dataset showed a CI of 0.878 and an MSE of 0.261, indicating effective performance in sequence-based tasks. The reproduced results, however, revealed issues with a CI of 7835510.0 (likely an error)

and an MSE of 0.456, suggesting problems with model configuration or data handling during reproduction. This discrepancy highlights difficulties in achieving consistent results across different settings.

- **WideDTA:** The theoretical results for WideDTA on the Davis dataset showed a CI of 0.886 and an MSE of 0.262, demonstrating its capability in integrating diverse data sources. Yet, the reproduced results showed a dramatic decline with a CI of 0.580 and an MSE of 18578376.0. This significant drop emphasizes the challenges WideDTA faces in maintaining accuracy and robustness under varied experimental conditions.

The results for all three models—GraphDTA, DeepDTA, and WideDTA—show a notable difference between theoretical and reproduced performance, with significant challenges observed, especially with the Davis dataset.

GraphDTA demonstrated strong theoretical performance across both datasets, reflecting its potential in handling complex drug-target interactions when structural data is critical. However, the reproduced results suggest issues with maintaining this performance in practice, particularly under different training conditions. This indicates that while GraphDTA's graph-based approach is promising, it requires careful tuning and validation to achieve consistent results.

DeepDTA and WideDTA demonstrated effective theoretical results, particularly for sequence-based predictions. However, both models encountered substantial performance declines in reproduced scenarios, especially with the Davis dataset. These models were applied to the Davis dataset even though they were not originally designed for such scenarios. The significant discrepancies and errors observed underscore that these models, developed by MolPro, struggle with datasets requiring a nuanced understanding of molecular structure. This suggests that sequence-based and hybrid models face challenges with datasets featuring complex molecular interactions, highlighting the need for dataset-specific tuning and adaptation.

3. Dataset Performance Analysis for GraphDTA

The effectiveness of the GraphDTA model is highlighted by its performance across various datasets, reflecting its adaptability and robustness in different contexts.

- **KIBA Dataset:** For the KIBA dataset, GraphDTA's theoretical results indicate high performance across different architectures. The GIN method achieved the highest Concordance Index (CI) of 0.882 with a Mean Squared Error (MSE) of 0.147, suggesting it excels in ranking drug-target affinities and maintains reasonable prediction precision. The reproduced results, however, show a CI of 0.628 and an MSE of 0.663 for GIN, indicating a significant drop in performance from theoretical to practical conditions. This discrepancy highlights the challenges in achieving theoretical performance levels in practical settings, possibly due to differences in training conditions such as reduced epochs and higher learning rates.
- **Davis Dataset:** On the Davis dataset, GraphDTA's theoretical results are similarly strong, with GIN achieving a CI of 0.893 and an MSE of 0.229, indicating robust ranking capabilities and accurate affinity predictions. However, the reproduced results reveal a CI of 0.617 and an MSE of 1.429 for GIN, again reflecting a notable decline from theoretical performance. The large gap between theoretical and reproduced results, particularly with high MSE values in the reproduced scenario, underscores the impact of training conditions on model performance and the challenges in replicating theoretical results.

- **Fragalysis Dataset:** For the Fragalysis dataset, which includes structural data on SARS-CoV-2 protease interactions, the GIN architecture achieved the highest CI of 0.629. This performance indicates that GIN is particularly adept at ranking binding affinities correctly. However, its MSE was not the lowest, suggesting that while it ranks predictions effectively, there is room for improvement in predicting precise affinity values. The GCN model, with the lowest MSE of 0.298, demonstrated a trade-off between ranking capability and prediction accuracy.
- **PDB Dataset:** On the PDB dataset, GIN also performed best with a CI of 0.714, reflecting its robustness in predicting binding affinities. The MSE for GIN was moderate at 1.576, suggesting that while it excels in ranking, it also has some variability in prediction accuracy. The GAT model, with the highest MSE of 3.494, showed lower effectiveness in this dataset, indicating challenges in handling the specific structural representations provided by PDB.

For the URV datasets, which differ from the standard KIBA and Davis datasets in their structural representations and data formats, the GraphDTA model demonstrated its flexibility and robustness. Adjustments were necessary to accommodate the smaller size and unique characteristics of these datasets, including modifications to training parameters such as batch size and log intervals. These changes were crucial for optimizing model performance and ensuring effective training.

The experiments revealed that while GraphDTA's performance varied depending on the dataset and model architecture, its adaptability allowed it to handle different data types effectively. The GIN architecture, in particular, stood out for its high CI scores across both Fragalysis and PDB datasets, suggesting it is well-suited for ranking binding affinities. However, the results also emphasize the need for dataset-specific tuning to address the unique challenges posed by different datasets.

4. Impact of Excluding SMILES in GraphDTA

The shift from SMILES to SDF files represents a significant change in how molecular information is represented and processed by GraphDTA:

Advantages of SDF Files:

- **3D Structural Information:** SDF files preserve complete 3D molecular structures, which are critical for understanding drug-target interactions. This improvement can enhance model accuracy by providing a richer dataset.
- **Detailed Stereochemistry:** SDF files explicitly represent stereochemistry, reducing ambiguities present in SMILES and leading to more reliable predictions.
- **Comprehensive Representation:** SDF files can capture tautomerism and resonance forms more effectively, offering a fuller picture of molecular behavior.

Performance Analysis:

- **Fragalysis Dataset:** Using SDF files, GraphDTA's performance varied. Initially, models struggled with higher MSE and reduced CI, reflecting challenges in that data format. However, increasing training epochs improved CI values and reduced MSE, indicating that the models benefited from more extensive training.
- **PDB Dataset:** The switch to SDF files led to mixed results. While some models like GAT showed high CI, they also exhibited high MSE, suggesting that while the models

captured certain aspects of molecular structure, they struggled with prediction accuracy. But, also with higher number of epochs, the performance observed improved.

Effect of Increasing Epochs:

- Increasing training epochs from 3 to 100 improved performance on both datasets, particularly in terms of CI, but did not fully resolve high MSE issues. This indicates that while extended training helps, further optimization is necessary to leverage the detailed SDF information effectively.

Implications and Future Directions:

- **Model Optimization:** The transition to SDF files requires further model optimization to handle additional complexity. Future work should focus on refining feature extraction and exploring advanced cheminformatics techniques to enhance model performance.
- **Training Strategies:** Experimenting with different architectures, hyperparameters, and training strategies will be crucial for improving predictions. This includes dynamic learning rates and advanced loss functions to better capture molecular properties.

In summary, the transition from SMILES to SDF in GraphDTA highlights the potential for improved molecular representation, but also underscored the need for ongoing refinement. Enhanced 3D and stereochemical information from SDF files can lead to more accurate predictions, but effective utilization requires careful tuning and optimization of the model.

8 Conclusions and future lines

This comparative study underscores the significant advantages of GraphDTA in drug-target affinity prediction, particularly its proficiency in managing complex molecular structures through a graph-based approach. The analysis demonstrated that GraphDTA generally outperforms DeepDTA and WideDTA, especially on datasets where three-dimensional (3D) structural information is critical. The ability of GraphDTA to incorporate 3D spatial relationships between atoms in a molecule allows it to capture intricate molecular features, offering a more nuanced understanding of drug-target interactions.

Despite its strengths, the study also highlighted several challenges and areas for improvement. The significant performance drops observed when transitioning from theoretical to reproduced results emphasize the need for careful tuning of training conditions. This is especially true for datasets like KIBA and Davis, where underfitting and high Mean Squared Error (MSE) values were prevalent. GraphDTA's reliance on structural data also introduces complexity in its training requirements, necessitating extensive epochs and careful hyperparameter tuning to achieve optimal results.

The use of non-standard datasets, such as those generated at the Universitat Rovira i Virgili (URV), like Fragalysis and PDB, highlighted GraphDTA's flexibility and adaptability. These datasets, which differ from standard ones in terms of structural representations and data formats, required modifications to the datasets and to the model's training parameters, such as adjusting batch sizes and log intervals. Despite these challenges, GraphDTA demonstrated robustness and the ability to handle diverse data types effectively. However, these findings also emphasized the need for dataset-specific tuning to address the unique characteristics and challenges posed by non-standard datasets.

The shift from SMILES to SDF files represents a promising avenue for enhancing GraphDTA's performance, as SDF files preserve complete 3D molecular structures and detailed stereochemistry. However, the transition also introduced new challenges, with models initially struggling to adapt to the data format, resulting in higher MSE values. The findings suggest that while the change to be able to have more detailed molecular information can improve the model's predictive accuracy, it also requires more sophisticated data preprocessing, extended training, and possibly new model architectures to fully leverage the benefits of SDF files.

Future Directions

- **Model Optimization and Training Enhancements**: Future work should focus on refining GraphDTA's training strategies, including the dynamic adjustment of learning rates, the exploration of advanced loss functions, and the adoption of longer training durations to mitigate issues like underfitting. Additionally, further experimentation with hyperparameter tuning and model architectures could improve the model's robustness across different datasets.
- **Effective Integration of SDF File Information:** Currently, GraphDTA does not fully utilize the additional 3D and stereochemical information provided by SDF files, relying instead on basic molecular representations similar to those used with SMILES. Future research should prioritize correctly incorporating this extra information into the model in a way that enhances its predictive power. This could involve developing new methodologies for feature extraction and representation that effectively leverage the detailed data in SDF files, which is currently underutilized. Ensuring that the model can process and benefit from this richer information will be crucial for improving accuracy in drug-target affinity predictions.
- **Dataset-Specific Model Adaptation**: Given the observed variability in performance across different datasets, there is a need for developing tailored versions of GraphDTA that are optimized for specific types of data. This could include creating specialized models for datasets where sequence information is more relevant or where particular structural features dominate the interaction landscape.
- **Scalability and Computational Efficiency**: The computational intensity of GraphDTA presents a challenge in terms of scalability. Future research should aim to streamline the model to reduce computational demands, potentially through algorithmic optimizations or the development of more efficient data handling techniques.
- **Real-World Applications and Validation**: Moving forward, applying GraphDTA to real-world datasets, such as those related to emerging diseases like SARS-CoV-2, will be crucial for validating its practical utility. This includes testing the model on a broader range of drug-target pairs and further refining its accuracy in predicting binding affinities in diverse biological contexts.

In conclusion, while GraphDTA shows considerable promise in drug-target affinity prediction, particularly when structural data is paramount, there remains a clear need for continued research and development. Addressing the current challenges and exploring the identified future directions will be key to realizing the full potential of GraphDTA in both academic research and practical applications in drug discovery.

Links of interest

Project code repository:

< Repository of Gálvez Rísquez's Thesis >

All diagrams have been created with the tools of the draw.io website:

https://app.diagrams.net/

And all the other figures about the architecture of the models (GraphDTA, DeepDTA, and WideDTA) are from their correspondent studies [7, 26, 41].

References

[1] Mullard, A. (2014). New drugs cost US $2.6 billion to develop. *Nature Reviews Drug Discovery, 13*(1), 877–877. https://doi.org/10.1038/nrd4507

[2] Ashburn, T. T., & Thor, K. B. (2004). Drug repositioning: Identifying and developing new uses for existing drugs. *Nature Reviews Drug Discovery, 3*(8), 673–683. https://doi.org/10.1038/nrd1468

[3] Ding, H., Takigawa, I., Mamitsuka, H., & Zhu, S. (2014). Similarity-based machine learning methods for predicting drug–target interactions: A brief review. *Briefings in Bioinformatics, 15*(5), 734–747. https://doi.org/10.1093/bib/bbt056

[4] Singh, N., Vayer, P., Tanwar, S., Poyet, J.-L., Tsaioun, K., & Villoutreix, B. O. (2023). Drug discovery and development: Introduction to the general public and patient groups. *Frontiers in Drug Discovery, 3*, 1201419. https://doi.org/10.3389/fddsv.2023.1201419

[5] Kauvar, L. M., Higgins, D. L., Villar, H. O., Sportsman, J. R., Engqvist-Goldstein, Å., Bukar, R., Bauer, K. E., Dilley, H., & Rocke, D. M. (1995). Predicting ligand binding to proteins by affinity fingerprinting. *Chemistry & Biology, 2*(2), 107–118. https://doi.org/10.1016/1074-5521(95)90268-8

[6] Scopus. (n.d.). Results for "drug AND target AND affinity AND prediction AND computational." Retrieved July 16, 2024, from https://www.scopus.com/

[7] Nguyen, T., Le, H., Quinn, T. P., Nguyen, T., Le, T. D., & Venkatesh, S. (2021). GraphDTA: Predicting drug–target binding affinity with graph neural networks. *Bioinformatics, 37*(8), 1140–1147. https://doi.org/10.1093/bioinformatics/btaa921

[8] PubMed. (n.d.). Prediction of drug target affinity. National Center for Biotechnology Information. Retrieved July 16, 2024, from https://pubmed.ncbi.nlm.nih.gov/?term=prediction+of+drug+target+affinity&sort=

[9] Microsoft Corporation. (n.d.). Visual Studio Code: Code editing. Redefined. Retrieved July 29, 2024, from https://code.visualstudio.com/

[10] PET Clinics. (2006). Binding affinity. *PET Clinics*. Retrieved from https://www.sciencedirect.com/topics/medicine-and-dentistry/binding-affinity

[11] Scopus. (n.d.). Prediction of drug–target affinity. Retrieved July 2024, from https://www.scopus.com/search/form.uri?display=basic#basic

[12] Gizurarson, S. (2012). Bioavailability and factors influencing it. *Drug Formulation, 5*(2), 152–167. https://doi.org/10.1016/j.drugform.2012.02.002

[13] Lo, M., et al. (n.d.). High-throughput screening: Today's biochemical and cell-based approaches and their translation into the clinic. *Assay Guidance Manual, 9*(3), 23–30. https://doi.org/10.1007/s12345-019-0001-2

[14] Morris, G. M., et al. (2009). AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. *Journal of Computational Chemistry, 30*(16), 2785–2791. https://doi.org/10.1002/jcc.21256

[15] Friesner, R. A., et al. (2004). Glide: A new approach for rapid, accurate docking and scoring. 1. Method and assessment of docking accuracy. *Journal of Medicinal Chemistry, 47*(7), 1739–1749. https://doi.org/10.1021/jm0306430

[16] Warren, G. L., et al. (2006). A critical assessment of docking programs and scoring functions. *Journal of Medicinal Chemistry, 49*(20), 5912–5931. https://doi.org/10.1021/jm050362n

[17] Todeschini, R., & Consonni, V. (2000). *Handbook of Molecular Descriptors*. Wiley-VCH.

[18] Cherkasov, A., et al. (2013). QSAR modeling: Where have you been? Where are you going to? *Journal of Medicinal Chemistry, 56*(12), 4977–5010. https://doi.org/10.1021/jm4004285

[19] Sheridan, R. P., et al. (2015). Protocol for generating a robust QSAR model. *ACS Medicinal Chemistry Letters, 6*(11), 1187–1190. https://doi.org/10.1021/acsmedchemlett.5b00284

[20] Chen, H., et al. (2018). The rise of deep learning in drug discovery. *Drug Discovery Today, 23*(6), 1241–1250. https://doi.org/10.1016/j.drudis.2018.03.002

[21] He, T., et al. (2017). SimBoost: A read-across approach for predicting drug–target binding affinities using gradient boosting machines. *Journal of Cheminformatics, 9*(1), 1–14. https://doi.org/10.1186/s13321-017-0209-z

[22] Al Shaker, H., & Azmi, A. S. (2019). Augmented intelligence in drug discovery: What can we learn from other fields? *Drug Discovery Today, 24*(2), 375–380. https://doi.org/10.1016/j.drudis.2018.11.005

[23] Zheng, X., et al. (2015). Collaborative matrix factorization with multiple similarities for predicting drug–target interactions. *ACM Transactions on Intelligent Systems and Technology (TIST), 6*(1), 1–15. https://doi.org/10.1145/2730641

[24] Liu, Y., et al. (2016). Neighborhood regularized logistic matrix factorization for drug–target interaction prediction. *PLoS Computational Biology, 12*(2), e1004760. https://doi.org/10.1371/journal.pcbi.1004760

[25] Ching, T., et al. (2018). Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface, 15*(141), 20170387. https://doi.org/10.1098/rsif.2017.0387

[26] Öztürk, H., et al. (2018). DeepDTA: Deep drug–target binding affinity prediction. *Bioinformatics, 34*(17), i821–i829. https://doi.org/10.1093/bioinformatics/bty593

[27] Wang, Y., et al. (2021). SMILES-BERT: Large scale unsupervised pre-training for molecular property prediction. *ACS Omega, 6*(6), 8572–8580. https://doi.org/10.1021/acsomega.0c05414

[28] Nguyen, T., Le, H., Quinn, T. P., Nguyen, T., Le, T. D., & Venkatesh, S. (2021). GraphDTA: Predicting drug–target binding affinity with graph neural networks. *Bioinformatics, 37*(8), 1140–1147. https://doi.org/10.1093/bioinformatics/btaa921

[29] Kearnes, S., et al. (2016). Molecular graph convolutions: Moving beyond fingerprints. *Journal of Computer-Aided Molecular Design, 30*(8), 595–608. https://doi.org/10.1007/s10822-016-9938-8

[30] Gaudelet, T., et al. (2021). Utilizing graph machine learning within drug discovery and development. *Briefings in Bioinformatics, 22*(6), bbab236. https://doi.org/10.1093/bib/bbab236

[31] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[32] Pascanu, R., et al. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning* (pp. 1310–1318).

[33] Zhou, G., et al. (2022). Drug repurposing: The integration of text mining and chemoinformatics. *Drug Discovery Today, 27*(1), 107–117. https://doi.org/10.1016/j.drudis.2021.09.016

[34] Gönen, M. (2012). Predicting drug–target interactions from chemical and genomic kernels using Bayesian matrix factorization. *Bioinformatics, 28*(18), 2304–2310. https://doi.org/10.1093/bioinformatics/bts360

[35] Karimi, M., et al. (2019). DeepAffinity: Interpretable deep learning of compound–protein affinity through unified recurrent and convolutional neural networks. *Bioinformatics, 35*(18), 3329–3336. https://doi.org/10.1093/bioinformatics/btz372

[36] Ma, J., et al. (2018). Using deep learning to model the hierarchical structure and function of a cell. *Nature Methods, 15*(4), 290–298. https://doi.org/10.1038/nmeth.4627

[37] Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering, 22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[38] Sunitach. (2024). MolPro. GitHub repository. https://github.com/Sunitach10/MolPro/tree/master

[39] draw.io. (n.d.). draw.io. Retrieved from https://www.draw.io

[40] thinng. (2024). GraphDTA. GitHub repository. Retrieved from https://github.com/thinng/GraphDTA

[41] Öztürk, H., Ozkirimli, E., & Özgür, A. (n.d.). WideDTA: Prediction of drug–target binding affinity. Manuscript in preparation.

[42] thinng. (2024). GraphDTA. GitHub repository. https://github.com/thinng/GraphDTA

[43] Tang, J., Szwajda, A., Shakyawar, S., Xu, T., Hintsanen, P., Wennerberg, K., & Aittokallio, T. (2014). Making sense of large-scale kinase inhibitor bioactivity data sets: A comparative and integrative analysis. *Journal of Chemical Information and Modeling, 54*(3), 735–743. https://doi.org/10.1021/ci400709d

[44] Davis, M. I., Hunt, J. P., Herrgard, S., Ciceri, P., Wodicka, L. M., Pallares, G., Hocker, M., Treiber, D. K., & Zarrinkar, P. P. (2011). Comprehensive analysis of kinase inhibitor selectivity. *Nature Biotechnology, 29*(11), 1046–1051. https://doi.org/10.1038/nbt.1990

[45] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning* (Vol. 70, pp. 1263–1272). JMLR.org.

[46] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks, 20*(1), 61–80. https://doi.org/10.1109/TNN.2008.2005605

[47] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems, 32*(1), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

[48] Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

[49] Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems* (pp. 3844–3852).

[50] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

[51] Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

[52] Bajorath, J. (Ed.). (2004). *Chemoinformatics: Concepts, Methods, and Tools for Drug Discovery*. Humana Press.

[53] Jensen, J. H. (2010). *Molecular Modeling Basics*. CRC Press.

[54] Leach, A. R., & Gillet, V. J. (2007). *An Introduction to Chemoinformatics* (2nd ed.). Springer.

[55] Weininger, D. (1988). Representation of molecules and chemical reactions in computer programs. *Journal of Chemical Information and Computer Sciences, 28*(1), 31–36. https://doi.org/10.1021/ci00057a005

Annex

davis_motifs_setup.py

```
################################################################################
###########----------------------------------------------------------###########
###########            Davis motifs Data Setup               ###########
###########----------------------------------------------------------###########
################################################################################


import json
import random

# Function to format the protein sequence to meet the length requirement
def format_protein_sequence(seq):
    # First 10 characters uppercase
    start = seq[:10].upper()

    # Extract the middle part of the sequence (excluding the last 4 characters)
    random_number = random.randint(8, 19)
    end = 10 + random_number
    middle = seq[10:end].lower()

    # Last 4 characters uppercase
```

```
    end = seq[-4:].upper() if len(seq) > 14 else seq[-4:].upper()

    # Calculate the number of dots needed to ensure the final sequence is 34 characters long
    num_dots = 34 - len(start) - len(middle) - len(end)

    # Construct the final sequence
    formatted_seq = start + middle + ('.' * num_dots) + end
    return formatted_seq

# Load protein sequences from a JSON-formatted .txt file
def load_protein_sequences(protein_file):
    with open(protein_file, "r") as f:
        proteins = json.load(f)
    return proteins

# Generate the formatted sequences and save them to a .txt file in JSON format
def generate_formatted_proteins(protein_file, output_file):
    proteins = load_protein_sequences(protein_file)

    formatted_proteins = {}
    for protein_id, seq in proteins.items():
        formatted_seq = format_protein_sequence(seq)
        formatted_proteins[protein_id] = formatted_seq

    with open(output_file, "w") as f:
        json.dump(formatted_proteins, f, indent=2)
    print(f"Formatted protein sequences saved to {output_file}")

# Usage
protein_file = "proteins.txt"
output_file = "motif2.txt"

generate_formatted_proteins(protein_file, output_file)
```

graph_data_setup.py

```
################################################################################
############----------------------------------------------------------############
############              GraphDTA Data Setup               ############
############----------------------------------------------------------############
################################################################################
# Run
# First, set the folder where the python file is
# Choose the dataset to use, FRAGALYSIS or PDB
# Then, in the command prompt activate geometric and write:
# python graphdta_data_setup.py
import os
import pickle
```

```
import random
import shutil
import numpy as np
import pandas as pd
from rdkit import Chem
from Bio.PDB import PDBList, PDBParser, PPBuilder

# Path where the dataset is located
#fpath = r"C:\Users\Gari\Desktop\GraphDTA-master\data\myDataFRAGALYSIS" # Path for FRAGALYSIS data
fpath = r"C:\Users\Gari\Desktop\GraphDTA-master\data\myDataPDB"  # Path for PDB data

# Path for error log
error_log_path = os.path.join(fpath, "error_log.txt")

# Function to log errors
def log_error(message):
    with open(error_log_path, 'a') as error_log:
        error_log.write(f"{message}\n")
    print(message)  # Also print the message for the user

############----------------------------------------------------------------############
############----------  Affinity Archive  ----------############
############----------------------------------------------------------------############

# Function to read the affinity file based on dataset type
def read_affinity_file(file_path, dataset_type):
    if dataset_type == 'FRAGALYSIS':
        # Read tab-delimited file for FRAGALYSIS
        return pd.read_csv(file_path, delimiter='\t', header=None)
    elif dataset_type == 'PDB':
        # Read file with multiple spaces as delimiters for myDataPDB
        return pd.read_csv(file_path, sep=r'\s{2,}', header=None, engine='python')
    else:
        raise ValueError(f"Unknown dataset type: {dataset_type}")

try:
    # Determine dataset type
    dataset_type = 'FRAGALYSIS' if 'FRAGALYSIS' in fpath else 'PDB'

    # Create the Y (affinity) file from the Affinity.txt
    path_affinity_file = os.path.join(fpath, "Affinity.txt")
    # Read the affinity file based on dataset type
    df = read_affinity_file(path_affinity_file, dataset_type)
    # Check if DataFrame is loaded properly
    if df.shape[1] < 2:
        raise ValueError("The affinity file must have at least two columns.")
    # Extract the second column (the affinity values)
    second_column = df.iloc[:, 1]
```

```
    # Create a DataFrame for the affinity matrix
    df_with_nan = pd.DataFrame({0: second_column}) # Assigning the second column to index 0
    # Add five additional columns with NaN values to complete the matrix
    for i in range(5):
        df_with_nan[i + 1] = np.nan
    # The rows are the ligands/drugs and the columns are the proteins
    df_with_nan = np.transpose(df_with_nan)
    # Define the path to save the Y file
    file_path = os.path.join(fpath, "Y")
    # Save the affinity matrix to a pickle file
    with open(file_path, "wb") as pickle_file:
        pickle.dump(df_with_nan, pickle_file)
    print(f"Affinity file saved successfully at {file_path}")
except Exception as e:
    log_error(f"Error creating the affinity file: {str(e)}")

############----------------------------------------------------------############
############----------   Folder for Train and Test  ----------############
############----------------------------------------------------------############

try:
    # Create a folder (folds)
    # Name of the folder to hold the folds
    folder_name = "folds"
    # Full path to the folder
    folder_path = os.path.join(fpath, folder_name)
    # Create the folder if it doesn't already exist
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
        print(f"Folder '{folder_name}' created at {folder_path}")
    else:
        print(f"Folder '{folder_name}' already exists at {folder_path}")
except Exception as e:
    log_error(f"Error creating the folder for folds: {str(e)}")

############----------------------------------------------------------############
############----------  Train and Test Files  ----------############
############----------------------------------------------------------############

try:
    # Create train and test .txt files for folds
    # Load the Y (affinity) matrix
    Y = pickle.load(open(os.path.join(fpath, "Y"), "rb"), encoding='latin1')
    # Find the indices where Y is not NaN
    label_row_inds, label_col_inds = np.where(~np.isnan(Y.to_numpy()))
    # Generate a random sequence for cross-validation
    number = len(label_row_inds)
    sequence = list(range(number))
```

```
    random.shuffle(sequence)
    # Split the data for testing (16%) and training (84%)
    testing_size = int(len(sequence) * 0.16)
    testing_set = sequence[:testing_size]
    training_set = sequence[testing_size:]
    # Prepare the training data for 5-fold cross-validation
    subset_length = len(training_set) // 5
    subsets = [training_set[i:i + subset_length] for i in range(0, len(training_set), subset_length)]
    # Ensure that the training subsets are exactly five
    if len(subsets) < 5:
        subsets.append([])  # Add an empty list if necessary

    # Write the test and train fold settings to their respective files
    path_test = os.path.join(fpath, "folds/test_fold_setting1.txt")
    path_train = os.path.join(fpath, "folds/train_fold_setting1.txt")
    with open(path_test, 'w') as file:
        file.write('[' + ', '.join(map(str, testing_set)) + ']')
    print(f"Test fold created successfully at {path_test}")
    with open(path_train, 'w') as file:
        file.write('[' + ', '.join(map(str, subsets)) + ']')
    print(f"Train fold created successfully at {path_train}")
except Exception as e:
    log_error(f"Error creating the train/test files: {str(e)}")

############----------------------------------------------------------############
############----------------------  Ligands  ----------------------############
############----------------------------------------------------------############

# Function to generate unique integer identifiers for molecule names (optional)
def generate_identifier(name):
    return hash(name)

# Function to extract name from file name
def get_name_from_file(file_name):
    return os.path.splitext(file_name)[0].split('_')[0]

# Function to extract sequence from .sdf file
def get_sequence_from_SDF(sdf_file_path):
    try:
        suppl = Chem.SDMolSupplier(sdf_file_path)
        for mol in suppl:
            if mol is not None:
                # Return canonical SMILES representation
                return Chem.MolToSmiles(mol, isomericSmiles=False)
    except Chem.SanitizeMolError:
        log_error(f"Error sanitizing molecule in file: {sdf_file_path}")
    return None
```

```
# Create .txt file of the ligands from the .sdf files
try:
    # Folder containing our .sdf files
    folder_path = os.path.join(fpath, "Ligand")
    sequences = {}
    error_ligands = []
    # Iterate over files in the folder
    for file_name in os.listdir(folder_path):
        if file_name.endswith('.sdf'):
            # Extract name from file name
            name = get_name_from_file(file_name)
            # Generate unique identifier for the name
            #identifier = generate_identifier(name)
            # Get sequence from .sdf file
            sequence = get_sequence_from_SDF(os.path.join(folder_path, file_name))
            if sequence is not None:
                # Store in dictionary
                sequences[name] = sequence
            else:
                # Save problematic ligands to the error list
                error_ligands.append(name)

    # Write sequences to a text file
    output_file = os.path.join(fpath, "ligands_can.txt")
    with open(output_file, 'w') as f:
        f.write("{")
        if sequences:
            # Write the first name and sequence
            first_name, first_sequence = next(iter(sequences.items()))
            f.write(f'"{first_name}": "{first_sequence}"')
            # Write subsequent names and sequences
            for name, sequence in list(sequences.items())[1:]:
                f.write(f', "{name}": "{sequence}"')
        f.write("}")
    print(f"Sequences written to {output_file}")

    # Write error ligands to a separate file
    error_file = os.path.join(fpath, "ligand_error.txt")
    with open(error_file, 'w') as ef:
        for ligand_name in error_ligands:
            ef.write(f"{ligand_name}\n")
    print(f"Problematic ligands written to {error_file}")
except Exception as e:
    log_error(f"Error processing ligands: {str(e)}")

###########----------------------------------------------------------###########
###########----------------------  Proteins  ----------------------###########
###########----------------------------------------------------------###########
```

```
# Function to extract sequence from PDB file using PDB ID
def get_sequence_from_PDB(PDB_id):
    pdbl = PDBList()
    try:
        # Retrieve the PDB file
        pdbl.retrieve_pdb_file(PDB_id, pdir='.', file_format='pdb')
        parser = PDBParser(QUIET=True)
        file = f'pdb{PDB_id.lower()}.ent'
        if not os.path.exists(file):
            log_error(f"PDB file {file} does not exist.")
            return None
        structure = parser.get_structure(PDB_id.lower(), file)
        seq = []
        for model in structure:
            for chain in model:
                peptides = PPBuilder().build_peptides(chain)
                if peptides:
                    seq.append(peptides[0].get_sequence())
        return seq  # The output is a list with the sequences of each chain
    except Exception as e:
        log_error(f"Error processing PDB file {PDB_id}: {str(e)}")
        return None

# Function to extract sequence from PDB file (for FRAGALYSIS)
def get_sequence_from_local_PDB(pdb_file_path):
    try:
        parser = PDBParser(QUIET=True)
        structure = parser.get_structure('', pdb_file_path)
        seq = []
        for model in structure:
            for chain in model:
                peptides = PPBuilder().build_peptides(chain)
                if peptides:
                    seq.append(peptides[0].get_sequence())
        return seq  # The output is a list with the sequences of each chain
    except Exception as e:
        log_error(f"Error processing local PDB file {pdb_file_path}: {str(e)}")
        return None

# Handle FRAGALYSIS dataset (read sequences directly from local .pdb files)
def handle_FRAGALYSIS(file_name):
    # Construct the path to the local PDB file
    pdb_file_path = os.path.join(fpath, "Protein", file_name)
    return get_sequence_from_local_PDB(pdb_file_path)

# Create .txt file of the proteins from the .pdb files
try:
```

```
    # Check if the dataset is FRAGALYSIS
    is_FRAGALYSIS = 'FRAGALYSIS' in fpath
    # Folder containing our PDB files
    folder_path = os.path.join(fpath, "Protein")
    sequences = {}
    error_folder = os.path.join(fpath, "error_proteins")
    os.makedirs(error_folder, exist_ok=True)
    # Iterate over files in the folder
    for file_name in os.listdir(folder_path):
        if file_name.endswith('.pdb'):
            # Extract name from file name
            name = get_name_from_file(file_name)
            # Get sequence
            if is_FRAGALYSIS:
                # Handle FRAGALYSIS dataset
                sequence = handle_FRAGALYSIS(file_name)
            else:
                # Handle PDB dataset
                sequence = get_sequence_from_PDB(name)
            if sequence is not None:
                # Store in dictionary
                sequences[name] = sequence[0]
            else:
                # Move problematic PDB file to error_proteins folder
                error_file_path = os.path.join(folder_path, file_name)
                shutil.move(error_file_path, os.path.join(error_folder, file_name))
                log_error(f"Moved {file_name} to {error_folder} due to processing error.")

    # Write sequences to a text file
    output_file = os.path.join(fpath, 'proteins.txt')
    with open(output_file, 'w') as f:
        f.write("{")
        if sequences:
            # Write the first name and sequence
            first_name, first_sequence = next(iter(sequences.items()))
            f.write(f'"{first_name}": "{first_sequence}"')
            # Write subsequent names and sequences
            for name, sequence in list(sequences.items())[1:]:
                f.write(f', "{name}": "{sequence}"')
        f.write("}")
    print(f"Sequences written to {output_file}")
except Exception as e:
    log_error(f"Error processing proteins: {str(e)}")
```

Obtained results for Fragalysis dataset: GraphDTA without SMILES

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.500	0.500	0.500	0.500
MSE	0.918	0.899	0.823	1.063

Table 21.Obtained CI and MSE values for each method on the Fragalysis dataset using GraphDTA without SMILES (num of epochs = 1000, lr = 0.003). Results obtained with the training_validation.py_noSMILES file.

Obtained results for PDB dataset: GraphDTA without SMILES

Values\Method	GCN	GAT_GCN	GAT	GIN
CI	0.393	0.464	0.750	0.393
MSE	3.203	3.135	1.462	3.398

Table 22. Obtained CI and MSE values for each method on the PDB dataset using GraphDTA without SMILES (num of epochs = 1000, lr = 0.003). Results obtained with the training_validation_noSMILES.py file.