**Explainable Graph Neural Networks for Off-Target Prediction of Kinase Inhibitor Drugs using GraphSAGE**

**Background**

* Off-target effects contribute significantly to drug toxicity and failure in clinical trials.
* GNNs can model biological interactions (e.g., drug-target, protein-protein).
* GraphSAGE offers inductive capability—useful for generalizing to unseen proteins.
* Model interpretability is critical for trust and biological relevance.

**Problem Statement**

Can we use an explainable Graph Neural Network architecture to predict and interpret off-target interactions of anti-cancer drugs?

### ****Objectives****

1. Build a GraphSAGE-based GNN to classify protein targets as on-target vs off-target.
2. Use model explainability (e.g., GNNExplainer) to highlight key features or substructures. Most current models are black-box or task-agnostic. I aim to highlight **specific subgraphs** (drug substructures + protein network paths) contributing to off-target binding adds significant interpretability.
3. Validate predictions against known off-target effects or literature.

## Model Architecture Overview

### ****1. Input Graph****

* **Nodes:** Drugs and Proteins.
* **Node Features:**
  + Drug: Morgan fingerprint (via RDKit) or learned embeddings.
  + Protein: One-hot amino acid, or pretrained embeddings (optional: ProtBERT).
* **Edges:**
  + Positive interactions (known DTI), negative sampled (no binding or unknown).
  + Optionally enrich with STRING PPI connections between proteins.

### ****Loss & Evaluation****

* Use **Binary Cross-Entropy Loss**
* Evaluation: **ROC-AUC**, **Precision-Recall**, **Confusion Matrix**

### ****Explainability****

Use GNNExplainer from PyTorch Geometric

Data Sources

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Source** | **Usage** |
| Drug-target pairs | [BindingDB](https://www.bindingdb.org/), [DrugBank](https://go.drugbank.com/) | Build labels for binding/non-binding |
| SMILES | DrugBank or ChEMBL | For graph conversion via RDKit |
| Protein features | UniProt + embeddings or one-hot | For protein node features |
| Off-target validation | SIDER or literature | Post-hoc validation of predictions |

## ools Checklist

* **PyTorch Geometric (PyG)** or **DGL**
* **RDKit** (for SMILES to graphs)
* **scikit-learn** (metrics)
* **Captum / GNNExplainer** for interpretability
* **Matplotlib / Seaborn** for visualizations

## Deliverables (for MSc submission)

* Final code notebook or repo
* Evaluation plots (ROC, PR, attention maps)

Dataset size = 100 drugs

## **Why 100 drugs is doable**

* Most drug–target datasets (e.g., **BindingDB**, **DrugBank**, **ChEMBL**) have thousands of annotated interactions.
* If you:
  + Use only **known anti-cancer drugs** (e.g., from **DrugBank's ATC codes**),
  + Focus on **human proteins** only,
  + And assume ~5–50 known interactions per drug,

… then **100 drugs would give you about 1,000–5,000 drug–protein interactions**, which is manageable for:

* + Training a GraphSAGE model (with mini-batching),
  + Computing pairwise features or graphs,
  + Explaining selected predictions (you don't have to explain all).

## ⚠️ What to Watch Out For

### 1. ****Data preprocessing cost****

* Processing 100 SMILES → graphs = fast (~minutes with RDKit)
* Protein features: if you use pretrained embeddings (like **ProtBERT**, **ESM**, or **UniRep**), batch the API or use saved vectors.
* Curating negatives (non-binding) will take more time — use **random sampling or decoys** carefully.

### 2. ****Model training time****

* GraphSAGE is fast and scalable.
* With PyTorch Geometric and mini-batching, you can train on 1000s of pairs in **minutes to a couple hours** depending on hyperparameters.
* Keep your architecture shallow (2–3 layers), and avoid overfitting.

### 3. ****Explainability scale****

* You don't need to explain all predictions.
* Instead, pick **5–10 representative drugs**, show which proteins were predicted as off-targets, and explain those.
* That’s enough for a thesis and even publication.

## ✅ Strategies to Make It Work

|  |  |
| --- | --- |
| **Task** | **Strategy** |
| **Drug selection** | Filter 100 anti-cancer drugs from **DrugBank** using ATC code **L01** (antineoplastics) |
| **SMILES → graphs** | Use **RDKit** in batch mode |
| **Target proteins** | Use top 1000–2000 **human proteins** with known DTIs from **BindingDB/ChEMBL** |
| **Protein features** | Use **precomputed UniRep or ESM-1b embeddings**, or one-hot |
| **Negative samples** | Sample random protein pairs for each drug with no known interaction |
| **Model** | GraphSAGE with hidden size 64–128, trained in <1h on CPU or GPU |
| **Explainability** | Use **GNNExplainer** for only a subset (~5–10 predictions) |
| **Validation** | Cross-validate on a held-out set, or check off-targets using **SIDER** or **OFFSIDES** datasets |
| **Write-up** | Document selection pipeline, graphs, and 1–2 detailed explainability case studies |

📌 If you **parallelize preprocessing**, and **don’t overcomplicate the model**, it’s **very feasible**.

Dataset Strategy for Kinase Inhibitors

|  |  |
| --- | --- |
| **Step** | **What You Do** |
| 🔍 Identify kinase inhibitors | Scrape/download from ChEMBL, DrugBank, or PKIDB |
| 📄 Extract SMILES + names | Use ChEMBL API or DrugBank |
| 🔗 Cross-match with BindingDB | Match drugs to DrugBank IDs in BindingDB |
| 🧬 Filter for Homo sapiens only | Use "Organism of Target" = Homo sapiens |
| 🧪 Keep kinase targets only | Filter targets via UniProt or name includes "kinase" |
| ✅ Select 100 drugs | Ensure each has 1–5 human kinase targets |
| 📊 Add off-target info | Use SIDER or BindingDB bioactivity data |
| 🔄 Sample negatives | For each drug, sample 2–5 proteins it does **not** bind |
| 💡 Optional | Validate top predicted off-targets using kinase trees or literature |

Here's a plan tailored to **automatically fetch 100 kinase inhibitors with human kinase targets**:

## ⚙️ Script Overview: fetch\_kinase\_inhibitors\_dataset.py

This script will:

1. ✅ Query **ChEMBL** for drugs labeled as kinase inhibitors
2. ✅ Get their **canonical SMILES** and ChEMBL/DrugBank IDs
3. ✅ Match these drugs with **BindingDB** interactions
4. ✅ Filter to human-only, **kinase-only targets**
5. ✅ Output:
   * kinase\_drugs.csv – ID + SMILES
   * kinase\_dti.csv – drug–kinase interaction table
   * protein\_sequences.fasta – for ESM embeddings

# **Fetch Kinase Inhibitors from ChEMBL**

We’ll:

1. Query ChEMBL’s target classification for "Protein Kinase" targets.
2. Retrieve all **approved small molecule inhibitors** targeting them.
3. Collect their ChEMBL IDs and SMILES.

# ✅ **Step 2: Fetch SMILES for Each Kinase Inhibitor**

We'll:

1. Read step1\_kinase\_inhibitors\_raw.csv
2. Get **canonical SMILES** for each molecule\_chembl\_id using the ChEMBL API
3. Save:
   * chembl\_id
   * canonical\_smiles
   * (Optional) molecule\_name

## Goal of Step 3

From your Step 1 file (step1\_kinase\_inhibitors\_raw.csv), you already have:

* target\_chembl\_ids for kinase targets

Now we want to:

1. Map each target\_chembl\_id to its **UniProt accession** or **protein sequence**
2. Get the **FASTA sequence** for that target
3. Save it into a usable .csv or .fasta file

## Step 4 Overview: Choose an Embedding Strategy

| **Method** | **Pros** | **Cons** | **Code Availability** |
| --- | --- | --- | --- |
| ✅ **ESM-1b** (by Meta) | State-of-the-art, pre-trained on UniProt | GPU preferred, large model | HuggingFace + PyTorch |
| ✅ **UniRep** | Compact, good generalization | Slightly older, less expressive | Easy Python wrappers |
| 🟨 One-hot | Simple, fast, interpretable | Low expressive power | DIY with NumPy |

## What Are Embeddings?

**Embeddings** are **vector representations** of complex, structured inputs — like:

* A **protein sequence**
* A **chemical molecule**
* A **word**, sentence, or image

They're usually **dense** (i.e., every element has a value) and **learned from data**, capturing **semantic, functional, or structural meaning** in a numerical format.

## ✅ **Step 5: Generate Drug Graphs (from SMILES)**

This step is where you **turn each drug into a molecular graph**:

* **Nodes = atoms**
* **Edges = chemical bonds**
* **Node features = atom properties** (e.g., atomic number, aromaticity, degree)
* Saved as: **graph objects** or exported as .pt, .pkl, or .npz files

## 🔧 What You’ll Do in Step 5

### Input:

* molecule\_chembl\_id
* SMILES strings from Step 2

### Output:

* A graph (per drug) with:
  + x: node features (atoms)
  + edge\_index: connectivity
  + (optionally) edge\_attr: bond types

### 🧪 You’ll Use:

* [RDKit](https://www.rdkit.org/) for SMILES parsing
* [PyTorch Geometric](https://pytorch-geometric.readthedocs.io/) (or DGL if preferred) for graph creation

**Step 6: Create Drug–Target Pair Dataset**

### ✅ What We’ll Do:

Build a dataset of samples where each row is:

scss

CopyEdit

(drug\_graph, target\_embedding) → binding\_label

### 📦 Input:

* Molecular graphs (from data/graphs/)
* Protein embeddings (from Step 4 one-hot: step4\_protein\_onehot.csv)
* Activity file (from Step 1: step1\_kinase\_inhibitors\_raw.csv)
  + Includes: molecule\_chembl\_id, target\_chembl\_id, activity\_value

### 🏷️ Labeling Strategy:

We’ll convert **IC50 / Ki / Kd** values into binary binding labels:

python

CopyEdit

label = 1 if activity\_value < 1000 else 0 # Example threshold (1 µM)

You can adjust the threshold depending on stringency.

## ✅ What the Final Dataset Looks Like

| **drug\_id** | **target\_id** | **label** |
| --- | --- | --- |
| CHEMBL12345 | CHEMBL1862 | 1 |
| CHEMBL98765 | CHEMBL1824 | 0 |

The actual model will:

* Load drug\_graph from file
* Load target\_embedding from CSV
* Use label to train

✅ **Successfully created the training pairs dataset** with 10,584 drug-target pairs

The dataset now contains:

* drug\_id: Molecule ChEMBL IDs (matching the molecular graphs)
* target\_id: Target ChEMBL IDs (matching the protein embeddings)
* label: Binary labels (1 for active/binding, 0 for inactive)

The data is now properly aligned across all three components:

1. Molecular graphs (from step 5)
2. Protein embeddings (from step 4)
3. Drug-target pairs (from step 6)

Perfect, Kevin. Now let’s enter **Step 7: Train a GraphSAGE model** that takes:

mathematica

CopyEdit

(Drug Graph + Protein Embedding) → Binding Probability

## 🧠 Step 7 Overview

### ✅ What You’ll Do:

1. **Define a custom PyTorch Geometric Dataset class**
   * Loads: .pt drug graph + protein embedding + label
2. **Build the model:**
   * GraphSAGE to encode the drug graph
   * MLP (fully connected layers) to combine drug & protein
3. **Train and evaluate** the model on your dataset

Drug Graph (GNN) → Drug embedding (128D)

Target Embedding (one-hot or ESM) → 1D vector

→ [Concat] → FC → FC → Sigmoid → Probability

Recap:

| **File** | **Purpose** |
| --- | --- |

|  |  |
| --- | --- |
| data/graphs/CHEMBLxxxx.pt | Drug graph data |

|  |  |
| --- | --- |
| data/step4\_protein\_onehot.csv | Protein features |

|  |  |
| --- | --- |
| data/step6\_training\_pairs.csv | Drug-target-label triplets |

## **Key Improvements Made:**

### 1. ****ROC-AUC Evaluation**** 📊

* Added from sklearn.metrics import roc\_auc\_score
* Updated evaluate() function to collect all predictions and labels
* Calculates ROC-AUC instead of simple accuracy
* Better for imbalanced datasets (AUC = 1.0 = perfect, AUC = 0.5 = random)

### 2. ****Enhanced Model Architecture**** 🧠

* Added a **third GraphSAGE layer** (conv3) for deeper graph learning
* Integrated **dropout (p=0.3)** after each GNN layer and MLP layer
* Prevents overfitting on your smaller dataset
* Better neighborhood coverage with multi-hop message passing

### 3. ****Improved Training Loop**** 🚀

* Better formatted output with epoch numbering (01, 02, etc.)
* Clear metric explanations in console output
* Professional training progress display

## **Why These Changes Matter:**

| **Feature** | **Benefit** | **Publication Value** |
| --- | --- | --- |
| **ROC-AUC** | Robust on imbalanced data | ✅ Required standard |
| **Dropout** | Prevents overfitting | ✅ Best practice |
| **3 GNN Layers** | Better graph representation | ✅ Enhanced performance |

## **Expected Improvements:**

* More stable training due to dropout regularization
* Better evaluation metric that handles class imbalance
* Deeper graph understanding with additional GraphSAGE layer
* Professional output formatting for monitoring progress

## **Next Steps You Might Consider:**

1. **Early Stopping** - Stop when validation AUC plateaus
2. **Class Weighting** - Handle severe imbalance if present
3. **Model Saving** - Save best performing model
4. **Hyperparameter Tuning** - Optimize dropout rate, learning rate, etc.

## 🧠 Summary

| **What** | **File/Folder** | **Needed for** |
| --- | --- | --- |
| Drug graphs | data/graphs/\*.pt | Drug structure input (GNN) |
| Protein features | data/step4\_protein\_onehot.csv | Target embedding input |
| DTI labels | data/step6\_training\_pairs.csv | Supervised learning (label) |

Once those are loaded in Colab, you're ready to:

👉 Train your GraphSAGE model with dropout & ROC-AUC  
👉 Monitor performance  
👉 Save model checkpoints to Google Drive

**Step 8: Explain model predictions** using **GNNExplainer**. This step is key to making your model interpretable, and it’s especially important if you're aiming to **publish** or impress a review committee.

## Goal

**Explain why your GraphSAGE model predicted a particular drug–target interaction.**

We'll use **GNNExplainer** to highlight:

* Important atoms (nodes) and bonds (edges) in the drug graph.
* How much they contributed to the prediction.
* Potential biological reasoning behind off-targets

## What You'll Need

| **Requirement** | **Purpose** |
| --- | --- |
| Trained model | The DrugTargetModel you trained in step 7 |
| Drug graph | From data/graphs/\*.pt |
| Protein vector | From step4\_protein\_onehot.csv |
| Prediction output | e.g., high-probability prediction for a non-binding (off-target) case |

**GNNExplainer is not part of training**, it’s part of **post-hoc explainability**, i.e., it runs after the model is trained.

these are the next steps:

Step 9: Evaluation & Results Summary

💡 What to do:

Aggregate your performance metrics:

GraphSAGE AUC, Accuracy, Loss

Baseline vs. Improved models

Summarize explainability findings (e.g., "GNNExplainer highlighted residues known to be in ATP-binding domains")

Include:

Confusion matrix

ROC curves

Bar charts comparing models (you already have one)

🛠️ Optional:

Add Precision-Recall curves (helpful if dataset is imbalanced)

Show example predictions with explanations (esp. for 2–3 drugs)

📄 Output:

results\_summary.ipynb

final\_results.json

plots/roc\_curve.png, plots/explainability\_case\_X.png, etc.

📝 Step 10: Thesis or Manuscript Writing

🧠 Structure:

Introduction

Drug off-target problem

Graph learning + explainability for biomedicine

Methods

Dataset collection (Step 1–4)

Graph construction & model architecture (Step 5–7)

Explainability (Step 8)

Results

Performance metrics

Explanation case studies

Discussion

Why GraphSAGE worked (or didn't)

Biological relevance of explanations

Limitations (e.g., negative sampling assumptions)

Conclusion

Key takeaway: structure-aware explainable GNNs help identify off-targets

Future Work

Better embeddings, more drugs, clinical data, model ensembles

Appendix

Detailed hyperparameters, training logs

📘 Output: MSc\_Thesis\_DrugOffTarget\_GNN\_Explainability.docx

o get those exact metrics, you would need to:

1. Load [best\_improved\_graphsage\_model.pth](vscode-file://vscode-app/c:/Users/FEL_BA_01/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ")
2. Apply threshold optimization with threshold = 0.600
3. This will give you the 0.8859 AUC and 0.8101 accuracy performance

The model file itself contains the same weights as the Improved GraphSAGE - the performance boost comes from the optimized decision threshold, not different model weights.

explainability/biological\_validation.py`: Biological relevance assessment