Predictive Graph Neural Network for Identifying HIV Inhibitors Using Weighted Graphs of Molecular Structures

Razi Rachman Widyadhana - 13523004[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*rr.widyadhana@gmail.com*, *13523004@std.stei.itb.ac.id*

***Abstract*—Human Immunodeficiency Virus (HIV) remains the world's number one killer of infectious diseases. The spread of HIV is growing rapidly and affects women, teenagers, and children. By its popularity, machine learning has revolutionized to tackle the time-consuming discovery of new treatments. This paper reveals how GNNs not only identify an HIV inhibitor by its molecular structure, but also highlight the significant role that weighted graph plays in improving the accuracy of the predictions. This paper offers new possibilities for healthcare industry developments through machine learning applications.**

***Keywords*—Graph Neural Network, HIV Inhibitor, Weighted Graph, Molecular Structure.**

I. Introduction

Human Immunodeficiency Virus (HIV) remains the world's number one killer of infectious diseases. HIV attacks cells that support the body in fighting the infection, making a person more vulnerable to other infections and diseases. If left untreated, HIV leads to Acquired Immunodeficiency Syndrome (AIDS). It is the late stage of HIV infection that happens when the virus badly damages the body's immune system [1].

As blood transfusions, usage of needles, and sexual intercourse increased, the spread of HIV is growing rapidly and affects women, teenagers, and children. By the end of 2017, the World Health Organization (WHO) reported that about 36.9 million people were living with HIV/AIDS, 940,000 deaths due to HIV, and newly 1.8 million people were infected with HIV or about 5,000 new infections per day.

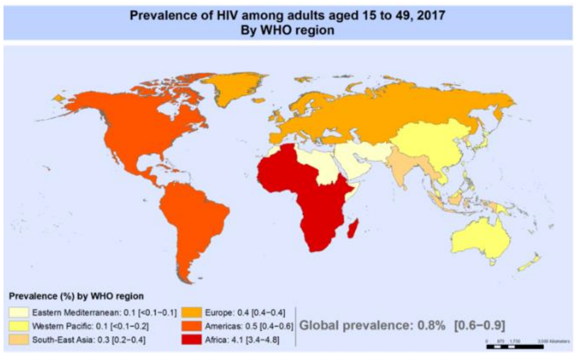

Prevalence of HIV among adults aged 15 to 49, 2017
By WHO region
Prevalence (%) by WHO region
Eastern Mediterranean: 0.1 [<0.1–0.1]
Western Pacific: 0.1 [<0.1–0.2]
South-East Asia: 0.3 [0.2–0.4]
Europe: 0.4 [0.4–0.4]
Americas: 0.5 [0.4–0.6]
Africa: 4.1 [3.4–4.8]
Global prevalence: 0.8% [0.6–0.9]

Fig. 1. HIV case reports in 2017 by WHO. Adapted from [2].

The human body cannot get rid of HIV and no effective HIV cure exists. Luckily, effective treatment with HIV medicine containing HIV inhibitors is available to help control the virus. This treatment is known as antiretroviral therapy (ART). ART is the best treatment method for preventing the progression of HIV to AIDS [3].

The exponential growth of technology influences the health industry in medicine development. By its popularity, machine learning has revolutionized to tackle the time-consuming discovery of new treatments. This paper utilizes Graph Neural Networks (GNNs), known for their accuracy in learning on graph-structured. Since HIV inhibitors are molecular structures that can be naturally represented as weighted graphs, GNNs are particularly well-suited for modelling their properties.

Therefore, this paper reveals how GNNs not only identify an HIV inhibitor by its molecular structure, but also highlight the significant role that weighted graph plays in improving the accuracy of the predictions.

II. Theoretical Foundations

A. Graph

Graphs are a form of data structure that consists of vertices and edges. The graph data structure has the notation G(V,E), where V is the set of vertices and E is the set of edges [4]. Graph theory is the branch of discrete mathematics that studies the properties and applications of graphs. In general, graphs are used to represent discrete objects and the relationships between them. According to the history of graphs, graphs have the purpose of visualizing abstract objects.

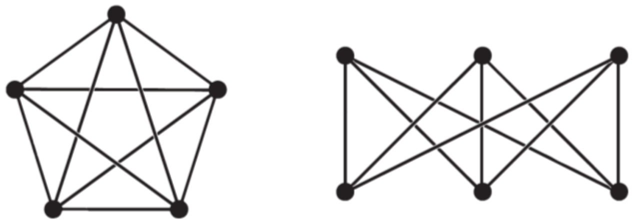


Fig. 2. Examples of Graph. Adapted from [4].

Each vertex of a graph has something called a degree. The degree of a vertex is the number of edges connected to that vertex. The degree of a vertex has the notation as a vertex. A vertex can be a remote vertex if the vertex has zero degree or no edge connected to the vertex.

Based on the Handshake Lemma, the number of degrees of all vertices in a graph is even, twice the number of edges. As a result of the Handshake lemma, it can be concluded that for any graph G, the number of vertices with odd degrees is always even. Based on this conclusion, a graph cannot have an odd number of vertices of odd degree.

Graphs can be divided into two types based on their direction, directed graph and undirected graph. an undirected graph is a type of graph in which all edges are bidirectional. This implies that the edges do not possess any inherent directionality.

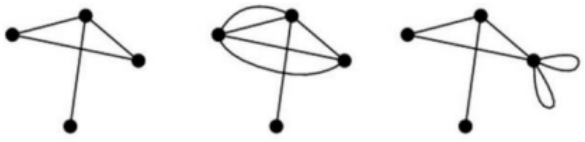


Fig. 3. Examples of Undirected Graphs. Adapted from [4].

In contrast, a directed graph consists of a sequence of vertices connected by directed edges, with each edge pointing from one vertex to its successor. A directed path excludes any repeated edges. If the path has no repeated vertices, it qualifies as a simple path. When the first and last vertices in a directed path match, and at least one edge connects them, this path becomes a directed cycle. A directed graph qualifies as a directed acyclic graph if it lacks directed cycles.

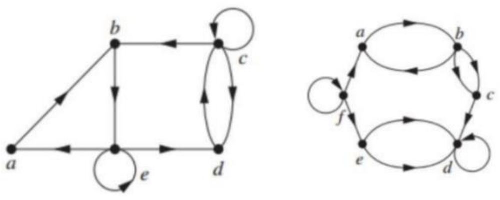


Fig. 4. Examples of Directed Graphs. Adapted from [4].

Graphs can also be divided into types based on whether or not weights are assigned. A weighted graph is a graph with each vertex or edge assigned to a number. This number is known as the weight of the graph. The weights can represent various concepts depending on the problem at hand.

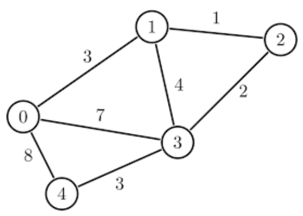


Fig. 5. Weighted Graph. Adapted from [5].

Graphs have several terminologies, including neighbor, side, path, circuit, and upagraph. Two vertices can be said to be connected or neighboring if there is an edge connecting them. there is an edge that connects the two vertices. Furthermore, there is the term side-by-side which means that an edge $e = (v_j, v_k)$ can be said that edge e is adjacent to the vertex v_j and v_k.

A path is a sequence of vertices and edges that connect a start vertex and an end vertex. Terminology that has a close relationship with trajectories is a circuit, which is a path that starts and ends at the same vertex. Lastly, an upagraph is a part of a graph and an upagraph G_1 can be said to be a part of a graph G if the vertices and edges of G_1 are a subset of the set of vertices and edges of graph G.

B. Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are specific machine learning models for processing data that can be represented as vectors on all graph attributes (nodes, edges, global-context). Ideally, these vector representations will have some meaningful relationship to the original graph, which makes them perfect for tasks involving relationships and dependencies between entities in complex systems [6].

The architecture involves multiple graph convolution layers that aggregate information from neighbouring nodes, allowing the network to learn rich representations of the graph structure. These layers are followed by non-linear activation functions such as ReLU to introduce complexity and dropout layers to prevent overfitting (exceptionally well on training data but fails to generalize to unseen data). The final output layer generates predictions for certain tasks.

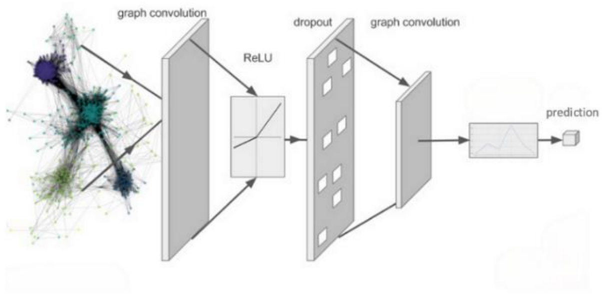
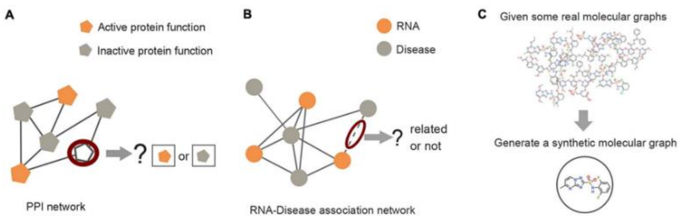


Fig. 6. Graph Neural Networks Architecture. Adapted from [7].

GNNs tasks can be broadly categorized into node-level, edge-level, and graph-level tasks. At the node level, tasks such as node classification aim to predict the properties of individual nodes, such as categorizing online users or items and anomaly detection in banking or networks. For link prediction, it identifies missing links between node pairs, like modelling interactions in social or biological networks. Graph-level tasks focus on learning representations of entire graphs, which are crucial in chemistry by representing molecules as graphs and predicting their properties. These various tasks highlight the flexibility of GNNs in tackling problems across different industries like healthcare, finance, and social networks within graph-structured data.


A
Active protein function
Inactive protein function
? or
PPI network
B
RNA
Disease
? related or not
RNA-Disease association network
C
Given some real molecular graphs
Generate a synthetic molecular graph
Fig. 7. Tasks of Graph Neural Networks (GNNs). Adapted from [8].

GNNs are closely associated with weighted graphs because of their capability to process and derive insights from these structures effectively. Weighted graphs provide additional information beyond simple connectivity by assigning weights, indicating the strength, distance, or other characteristics of the relationships between nodes and the nodes themselves. Integrating GNNs with weighted graphs allows these neural networks to exploit the richer information embedded in the graph structure.

C. Evaluation Metric

In evaluating the performance of a classification model, selecting appropriate metrics is essential to ensure that the predicted results are fit for purpose analysis. Some metrics have different interpretations in the context of evaluating multi-class classification methods. The relevant quantities for calculating the metrics for a binary class containing 0 (negative) and 1 (positive) are the four entries in the confusion matrix.

$$\mathrm{M} = \begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}$$

where TN denotes the number of correctly classified negative samples (True Negative), TP denotes the number of correctly classified positive samples (True Positive), FN denotes the number of samples incorrectly classified as negative (False Negative), and FP denotes the number of samples incorrectly classified as positive (False Positive). From those quantities, here are the metrics that are often used.

1) Precision: The precision denotes the proportion of the retrieved samples which are relevant and is calculated as the ratio between correctly classified samples and all samples assigned to that class.

$$Prec = \frac{TP}{TP + FP} \tag{2}$$

2) Recall: The recall, also known as the sensitivity or True Positive Rate (TPR), denotes the rate of positive samples correctly classified, and is calculated as the ratio between correctly classified positive samples and all samples assigned to the positive class.

$$Rec = \frac{TP}{TP + FN} \tag{3}$$

This metric known as being among the most important for medical studies, since it is desired to miss as few positive instances as possible, which translates to a high recall.

3) F1 score: The F1 score is the harmonic mean of precision and recall, meaning that it penalizes extreme values of either. This metric is not symmetric between the classes, it depends on which class is defined as positive and negative.

$$F1 = 2 \times \frac{Prec \times Rec}{Prec + Rec} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{4}$$

D. HIV Inhibitors

Approved antiretroviral (ARV) HIV medicines containing HIV inhibitors are divided into two main classes based on how each interferes with the HIV life cycle [9]. Experts advise combining the medicines to avoid creating medicine-resistant strains of HIV.

1) Block the replication process: Nucleoside Reverse Transcriptase Inhibitors (NRTIs) reverse transcriptase and reverse transcription prevents HIV from replicating. HIV uses reverse transcriptase to convert its RNA into DNA (reverse transcription). Similar to NRTIs, Non-nucleoside reverse Transcriptase Inhibitors (NNRTIs) bind to and later block reverse transcriptase, an enzyme HIV needs to make copies of itself. Protease Inhibitors (PIs) block protease. Lastly, Integrase Strand Transfer Inhibitors (INSTIs) stop HIV from making copies of itself by blocking a key protein that allows the virus to put its DNA into the healthy cell's DNA.

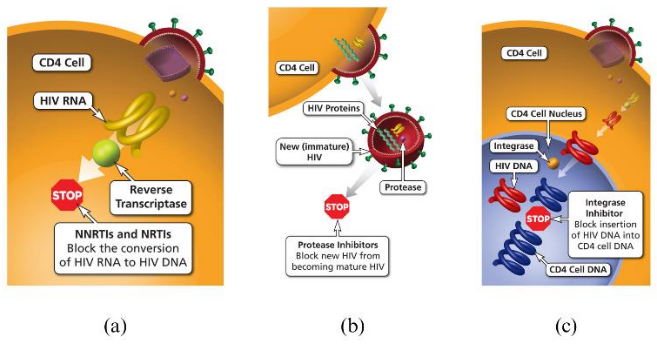

CD4 Cell
HIV RNA
STOP
Reverse Transcriptase
NNRTIs and NRTIs Block the conversion of HIV RNA to HIV DNA
CD4 Cell
HIV Proteins
New (immature) HIV
Protease
STOP
Protease Inhibitors Block new HIV from becoming mature HIV
CD4 Cell
CD4 Cell Nucleus
Integrase
HIV DNA
STOP
Integrase Inhibitor Block insertion of HIV DNA into CD4 cell DNA
CD4 Cell DNA
(a)
(b)
(c)

Fig. 8. Different inhibitors mechanism in fighting HIV, (a) NRTIs and NNRTIs, (b) PIs, and (c) INSTIs. Adapted from [9]

2) Prevent the cell entering process: Unlike NRTIs, NNRTIs, PIs, and INSTIs, which work on infected cells, fusion inhibitors block HIV from getting inside healthy cells.

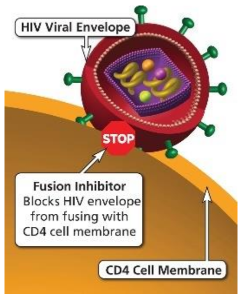

HIV Viral Envelope
STOP
Fusion Inhibitor Blocks HIV envelope from fusing with CD4 cell membrane
CD4 Cell Membrane

Fig. 9. Fusion Inhibitor mechanism in fighting HIV. Adapted from [9]

E. SMILES and Molecular Structure

Simplified Molecular-Input Line-System (SMILES) is a chemical notation used for information processing from modern chemistry to describe chemical structures with ASCII characters [10].

SMILES denotes a molecular structure as a graph and essentially the two-dimensional valence-oriented picture chemists draw to describe a molecule. This is an essential simplification of molecular structure. Extracting SMILES from a molecular structure contains the following four rules:

1) Atoms: Atoms are represented by their element symbols. All elements of a SMILES string are written in square brackets, like [Au] for elemental gold, with the exceptions of the organic subset, such as B, C, N, O, P, S, F, Cl, Br, and I. Attached hydrogens are implied in the absence of brackets, the following atomic symbols are valid SMILES notations.

TABLE I
VALID SMILES NOTATIONS FOR ATOMIC SYMBOLS

SMILES Notation	Description
C	Methane (CH_4)
N	Ammonia (NH_3)
O	Water (H_2O)
P	Phosphine (PH_3)
S	Hydrogen sulfide (H_2S)
Cl	Hydrogen chloride (HCl)

Explicit notation of hydrogen atoms occurs when they are attached to a non-organic subset. Attached hydrogens and formal charges are always specified inside brackets. The number of attached hydrogens is shown by the symbol H followed by an optional digit. Similarly, a formal charge is shown by one of the symbols + or -, followed by an optional digit. If unspecified, the number of attached hydrogens and charges is assumed to be zero for an atom inside the bracket.

TABLE II
SMILES NOTATIONS FOR ATTACHED HYDROGEN AND FORMAL CHARGES

SMILES Notation	Description
[H+]	Proton
[OH-]	Hydroxyl anion
[OH3+]	Hydronium cation
[Fe+2]	Iron (II) cation
[NH4+]	Ammonium cation

SMILES also recognizes constructions of the form [Fe+++] as synonymous with the form [Fe+3].

2) Bonds: Within the SMILES nomenclature, bonds may be, and usually are, omitted if they are either aromatic or single covalent bonds. Double bonds are represented with '=', and triple bonds are represented by '#'.

TABLE III
SMILES NOTATIONS FOR STRUCTURES WITH BONDS

SMILES Notation	Description
CC	Ethane (CH_3CH_3)
C=C	Ethylene (CH_2=CH_2)
COC	Dimethyl ether (CH_3OCH_3)
CCO	Ethanol (CH_3CH_2OH)
C=O	Formaldehyde (CH_2O)
O=C=O	Carbon dioxide (CO_2)
O=CO	Formic acid (HCOOH)
C#N	Hydrogen cyanide (HCN)
[H][H]	Molecular Hydrogen (H_2)

3) Branches: Branches are depicted in parentheses '()'. Branches can be nested or stacked. The structure's name is according to the convention of IUPAC (International Union of Pure and Applied Chemistry).

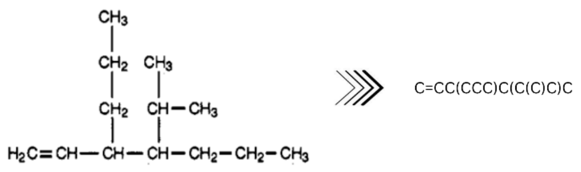


Fig. 10. SMILES of 3-propyl-4- isopropyl- 1 -heptene. Adapted from [10].

4) Cyclic Structures: Some molecules are in cyclic structures. They are converted *in-silico* to linear structures by breaking a single or aromatic bond within the cycles. For SMILES extraction, the broken bonds are denoted by writing a number right behind the formerly connecting elements. This leaves a connected noncyclic graph.

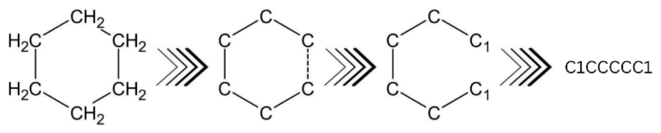


Fig. 11. SMILES of Cyclohexane, a cyclic structure. Adapted from [11].

5) Aromatic: Aromaticity is detected by applying an extended definition of Hückel's rule, defined as (1)

$$4n + 2\, pi\ electrons \iff aromatic \tag{1}$$

where n is a non-negative integer. If there is no n that fulfill the condition, then a planar ring molecule will not have aromaticity.

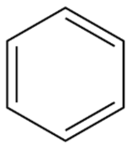


Fig. 12. Benzene, the most widely recognized aromatic compound with six delocalized π-electrons ($4n + 2, for\ n = 1$). Adapted from [11].

Aromaticity within SMILES is denoted by writing the atoms part of an aromatic cycle in lowercase letters.

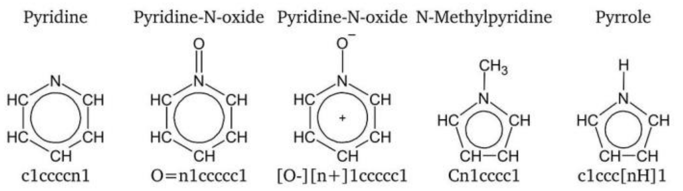


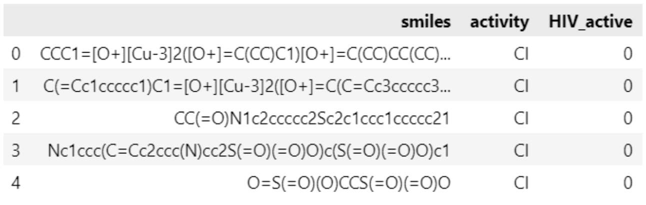
Fig. 13. Different instances of aromatic Nitrogen, whose SMILES are denoted by writing it in lowercase letters. Adapted from [11].

III. IMPLEMENTATION

The instruments used in this paper consist of two main components, namely hardware and software. The hardware specifications used are: Intel i7-8550U Processor, Intel UHD Graphics 620 GPU, and 8 GB RAM. Then, the software specifications used are Python 3.12.4 programming language and RDKit open source cheminformatics toolkit for generating graph weights from SMILES, as well as Jupyter Notebook and Kaggle for the program kernel.

A. Dataset Exploration

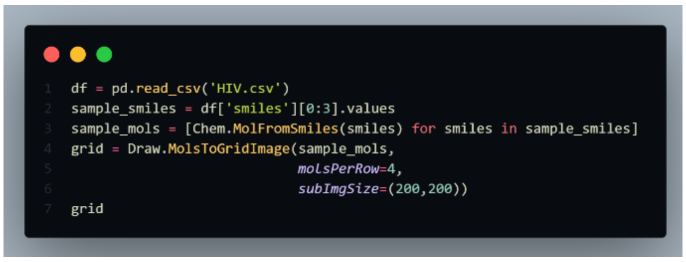
In this phase, the dataset was obtained from [12], which contains detailed information on 41,127 chemical structures. Each chemical structure is described under the SMILES format and their activities, evaluating the inhibitor activity (1) or the inactivity (0) of the chemical structures.



	smiles	activity	HIV_active
0	CCC1=[O+][Cu-3]2([O+]=C(CC)C1)[O+]=C(CC)CC(CC)...	CI	0
1	C(=Cc1ccccc1)C1=[O+][Cu-3]2([O+]=C(C=Cc3ccccc3...	CI	0
2	CC(=O)N1c2ccccc2Sc2c1ccc1ccccc21	CI	0
3	Nc1ccc(C=Cc2ccc(N)cc2S(=O)(=O)O)c(S(=O)(=O)O)c1	CI	0
4	O=S(=O)(O)CCS(=O)(=O)O	CI	0

Fig. 14. Samples of the HIV Dataset

For better understanding, the SMILES are visualized to their chemical structure form. Using RDkit tool to visualize the SMILES in their chemical structure form,



```python
df = pd.read_csv('HIV.csv')
sample_smiles = df['smiles'][0:3].values
sample_mols = [Chem.MolFromSmiles(smiles) for smiles in sample_smiles]
grid = Draw.MolsToGridImage(sample_mols,
                            molsPerRow=4,
                            subImgSize=(200,200))
grid
```

Fig. 15. Molecule Structure Visualization Algorithm in Python

Fig. 15. creates graph as chemical structure from selected samples with colors for each atoms.

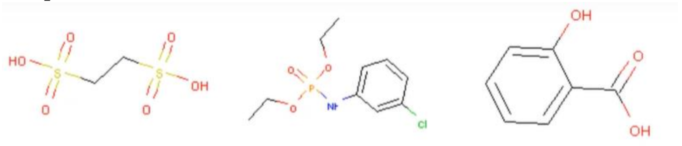


Fig. 16. Samples of Molecule Structure in the HIV dataset

The dataset was split into 80% training and 20% testing sets. The training set were oversampled for training the model to learn patterns and relationships with imbalanced class. Meanwhile, the testing set evaluates the model's performance by assessing its ability to generalize to unseen data. Visualization of the train set and test set activity distribution before oversample is shown in Fig. 17.

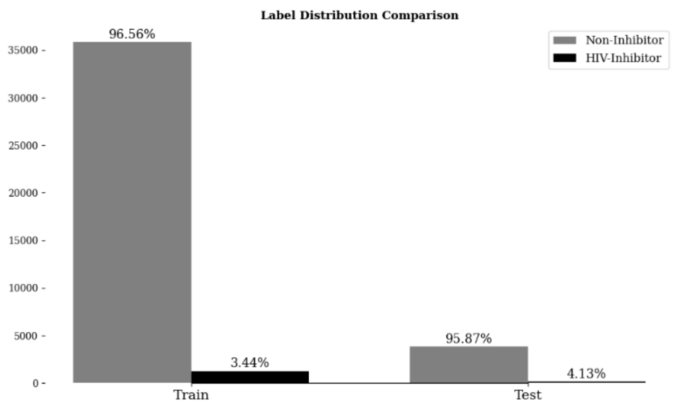
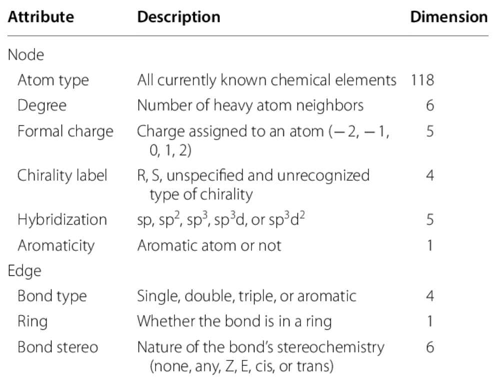


Fig. 17. Distribution of train set and test set of HIV dataset

B. Weight Extraction

After preparing the dataset, the next step is to extract weights for each edge in the graph. This paper utilizes additional libraries, namely Torch for creating embeddings for the graph's nodes/edges and RDKit for handling the molecular data and compute relevant properties.

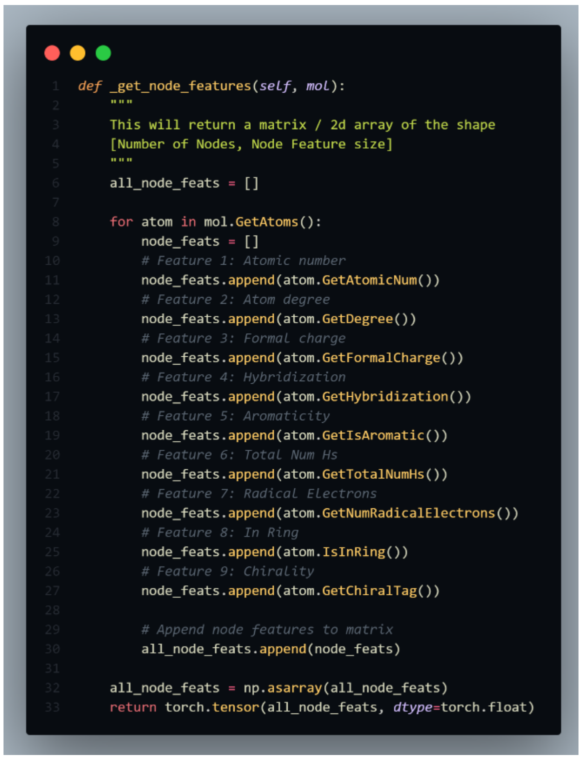
First, the Molecule class is created to represent molecular structures and store essential information about nodes and edges. The attributes extracted from the molecular data, as shown in Fig. 17., include node features such as atom type, degree, formal charge, chirality label, hybridization, and aromaticity. Edge features include bond type, whether the bond is part of a ring, and bond stereochemistry. These weights are crucial for analyzing and modeling the graph effectively.



Attribute	Description	Dimension
Node		
Atom type	All currently known chemical elements	118
Degree	Number of heavy atom neighbors	6
Formal charge	Charge assigned to an atom (−2, −1, 0, 1, 2)	5
Chirality label	R, S, unspecified and unrecognized type of chirality	4
Hybridization	sp, sp^2, sp^3, sp^3d, or sp^3d^2	5
Aromaticity	Aromatic atom or not	1
Edge		
Bond type	Single, double, triple, or aromatic	4
Ring	Whether the bond is in a ring	1
Bond stereo	Nature of the bond's stereochemistry (none, any, Z, E, cis, or trans)	6

Fig. 18. Attributes and its dimensions for node and edge features used in the Molecular Graph. Adapted from [13].

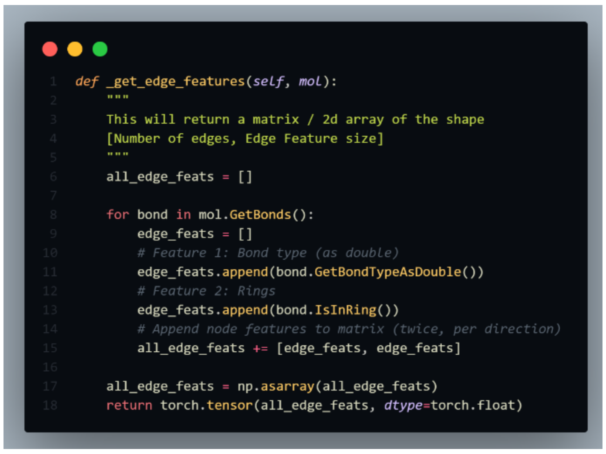
Inside the Molecule class, `_get_node_features` is defined to utilizes RDkit tool to extract the weights for node features arrays to the Graph as `torch` sensor.



```python
def _get_node_features(self, mol):
    """
    This will return a matrix / 2d array of the shape
    [Number of Nodes, Node Feature size]
    """
    all_node_feats = []

    for atom in mol.GetAtoms():
        node_feats = []
        # Feature 1: Atomic number
        node_feats.append(atom.GetAtomicNum())
        # Feature 2: Atom degree
        node_feats.append(atom.GetDegree())
        # Feature 3: Formal charge
        node_feats.append(atom.GetFormalCharge())
        # Feature 4: Hybridization
        node_feats.append(atom.GetHybridization())
        # Feature 5: Aromaticity
        node_feats.append(atom.GetIsAromatic())
        # Feature 6: Total Num Hs
        node_feats.append(atom.GetTotalNumHs())
        # Feature 7: Radical Electrons
        node_feats.append(atom.GetNumRadicalElectrons())
        # Feature 8: In Ring
        node_feats.append(atom.IsInRing())
        # Feature 9: Chirality
        node_feats.append(atom.GetChiralTag())

        # Append node features to matrix
        all_node_feats.append(node_feats)

    all_node_feats = np.asarray(all_node_feats)
    return torch.tensor(all_node_feats, dtype=torch.float)
```

Fig. 19. Node Weights Extraction Algorithm in Python.

Then, `_get_edge_features` is defined to utilizes RDkit tool to extract the weights for edge features arrays to the Graph as `torch` sensor.

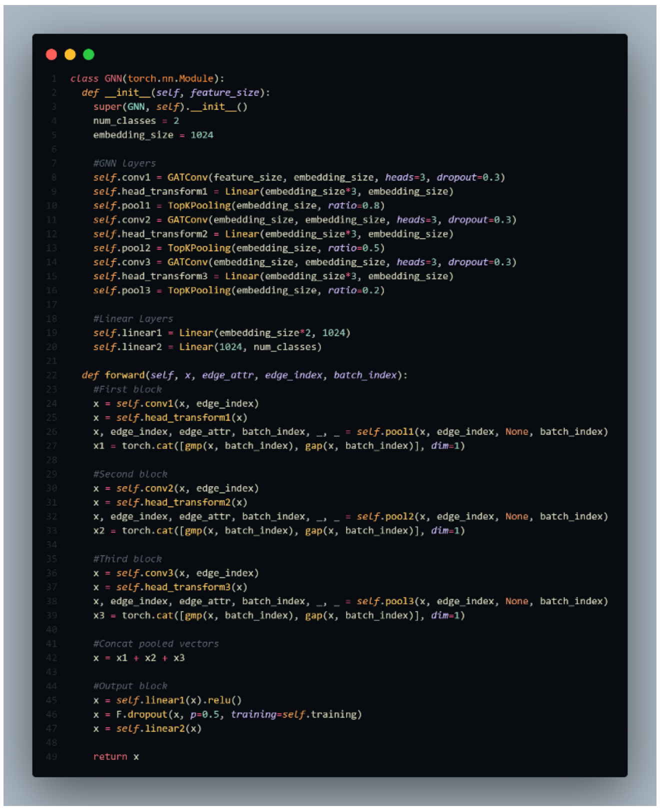


```python
def _get_edge_features(self, mol):
    """
    This will return a matrix / 2d array of the shape
    [Number of edges, Edge Feature size]
    """
    all_edge_feats = []

    for bond in mol.GetBonds():
        edge_feats = []
        # Feature 1: Bond type (as double)
        edge_feats.append(bond.GetBondTypeAsDouble())
        # Feature 2: Rings
        edge_feats.append(bond.IsInRing())
        # Append node features to matrix (twice, per direction)
        all_edge_feats += [edge_feats, edge_feats]

    all_edge_feats = np.asarray(all_edge_feats)
    return torch.tensor(all_edge_feats, dtype=torch.float)
```

Fig. 20. Edge Weights Extraction Algorithm in Python.

C. Graph-Level Classification

The first GNNs architecture employs Graph Attention Networks (GAT) to process graph-level classification tasks. GAT utilizes attention mechanisms to assign different importance scores to neighboring nodes, allowing the model to focus more on relevant connections within the graph.



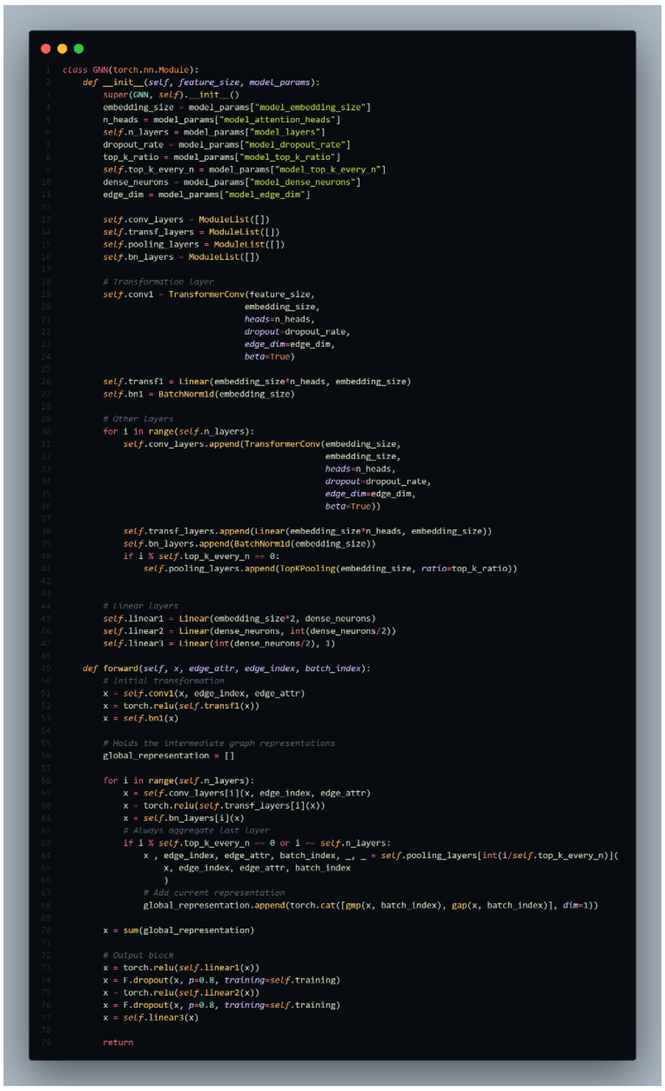
```python
class GNN(torch.nn.Module):
  def __init__(self, feature_size):
    super(GNN, self).__init__()
    num_classes = 2
    embedding_size = 1024

    #GNN layers
    self.conv1 = GATConv(feature_size, embedding_size, heads=3, dropout=0.3)
    self.head_transform1 = Linear(embedding_size*3, embedding_size)
    self.pool1 = TopKPooling(embedding_size, ratio=0.8)
    self.conv2 = GATConv(embedding_size, embedding_size, heads=3, dropout=0.3)
    self.head_transform2 = Linear(embedding_size*3, embedding_size)
    self.pool2 = TopKPooling(embedding_size, ratio=0.5)
    self.conv3 = GATConv(embedding_size, embedding_size, heads=3, dropout=0.3)
    self.head_transform3 = Linear(embedding_size*3, embedding_size)
    self.pool3 = TopKPooling(embedding_size, ratio=0.2)

    #Linear Layers
    self.linear1 = Linear(embedding_size*2, 1024)
    self.linear2 = Linear(1024, num_classes)

  def forward(self, x, edge_attr, edge_index, batch_index):
    #First block
    x = self.conv1(x, edge_index)
    x = self.head_transform1(x)
    x, edge_index, edge_attr, batch_index, _, _ = self.pool1(x, edge_index, None, batch_index)
    x1 = torch.cat([gmp(x, batch_index), gap(x, batch_index)], dim=1)

    #Second block
    x = self.conv2(x, edge_index)
    x = self.head_transform2(x)
    x, edge_index, edge_attr, batch_index, _, _ = self.pool2(x, edge_index, None, batch_index)
    x2 = torch.cat([gmp(x, batch_index), gap(x, batch_index)], dim=1)

    #Third block
    x = self.conv3(x, edge_index)
    x = self.head_transform3(x)
    x, edge_index, edge_attr, batch_index, _, _ = self.pool3(x, edge_index, None, batch_index)
    x3 = torch.cat([gmp(x, batch_index), gap(x, batch_index)], dim=1)

    #Concat pooled vectors
    x = x1 + x2 + x3

    #Output block
    x = self.linear1(x).relu()
    x = F.dropout(x, p=0.5, training=self.training)
    x = self.linear2(x)

    return x
```

Fig. 21. First version of GNNs using Convolutional Layers in Python.

It begins with an input layer that takes node features and processes them through three GAT convolutional layers, each featuring multi-head attention mechanisms with three heads and dropout regularization. Multi-head attention improves model by capturing diverse aspects of node relationships, while dropout prevents overfitting by randomly deactivating connections during training. These layers enhance the model's ability to capture complex relationships within the graph structure.

After each convolutional layer, linear transformations condense the multi-head outputs into a unified embedding size. In `forward` function, the outputs of the pooling layers are concatenated using both global mean pooling (GMP) and global attention pooling (GAP) to produce rich graph-level representations. GMP calculates the average of node features, while GAP uses attention weights to aggregate node features selectively. Finally, the concatenated outputs pass through two fully connected linear layers with dropout for regularization, concluding with a classification layer that predicts the target labels.

The second GNN architecture utilizes TransformerConv layers to leverage attention mechanisms and capture long-range dependencies within graphs. TransformerConv extends the traditional Transformer model, widely used in natural language processing, to graph data by incorporating edge features and multi-head attention for effective information propagation. The model initialization allows flexible configurations, such as embedding size, number of attention heads, dropout rates, and pooling ratios.

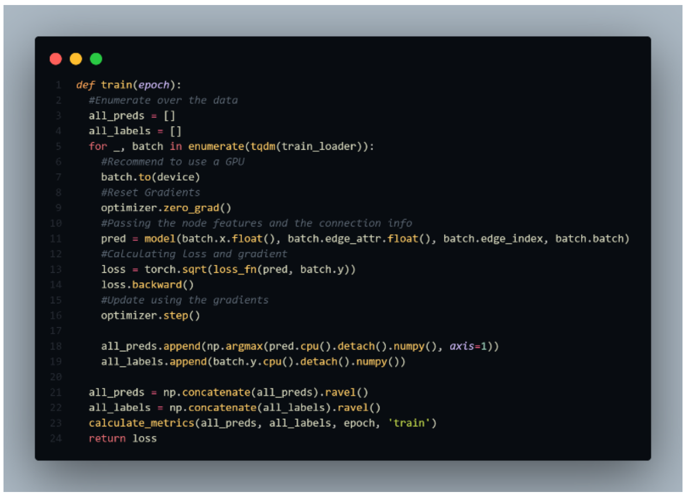


```python
class GNN(torch.nn.Module):
    def __init__(self, feature_size, model_params):
        super(GNN, self).__init__()
        embedding_size = model_params["model_embedding_size"]
        n_heads = model_params["model_attention_heads"]
        self.n_layers = model_params["model_layers"]
        dropout_rate = model_params["model_dropout_rate"]
        top_k_ratio = model_params["model_top_k_ratio"]
        self.top_k_every_n = model_params["model_top_k_every_n"]
        dense_neurons = model_params["model_dense_neurons"]
        edge_dim = model_params["model_edge_dim"]

        self.conv_layers = ModuleList([])
        self.transf_layers = ModuleList([])
        self.pooling_layers = ModuleList([])
        self.bn_layers = ModuleList([])

        # Transformation layer
        self.conv1 = TransformerConv(feature_size,
                                     embedding_size,
                                     heads=n_heads,
                                     dropout=dropout_rate,
                                     edge_dim=edge_dim,
                                     beta=True)

        self.transf1 = Linear(embedding_size*n_heads, embedding_size)
        self.bn1 = BatchNorm1d(embedding_size)

        # Other layers
        for i in range(self.n_layers):
            self.conv_layers.append(TransformerConv(embedding_size,
                                                    embedding_size,
                                                    heads=n_heads,
                                                    dropout=dropout_rate,
                                                    edge_dim=edge_dim,
                                                    beta=True))

            self.transf_layers.append(Linear(embedding_size*n_heads, embedding_size))
            self.bn_layers.append(BatchNorm1d(embedding_size))
            if i % self.top_k_every_n == 0:
                self.pooling_layers.append(TopKPooling(embedding_size, ratio=top_k_ratio))


        # Linear layers
        self.linear1 = Linear(embedding_size*2, dense_neurons)
        self.linear2 = Linear(dense_neurons, int(dense_neurons/2))
        self.linear3 = Linear(int(dense_neurons/2), 1)

    def forward(self, x, edge_attr, edge_index, batch_index):
        # Initial transformation
        x = self.conv1(x, edge_index, edge_attr)
        x = torch.relu(self.transf1(x))
        x = self.bn1(x)

        # Holds the intermediate graph representations
        global_representation = []

        for i in range(self.n_layers):
            x = self.conv_layers[i](x, edge_index, edge_attr)
            x = torch.relu(self.transf_layers[i](x))
            x = self.bn_layers[i](x)
            # Always aggregate last layer
            if i % self.top_k_every_n == 0 or i == self.n_layers:
                x , edge_index, edge_attr, batch_index, _, _ = self.pooling_layers[int(i/self.top_k_every_n)](
                    x, edge_index, edge_attr, batch_index
                    )
                # Add current representation
                global_representation.append(torch.cat([gmp(x, batch_index), gap(x, batch_index)], dim=1))

        x = sum(global_representation)

        # Output block
        x = torch.relu(self.linear1(x))
        x = F.dropout(x, p=0.8, training=self.training)
        x = torch.relu(self.linear2(x))
        x = F.dropout(x, p=0.8, training=self.training)
        x = self.linear3(x)

        return
```

Fig. 22. Second version of GNNs using Transformers in Python.

It starts with an initial TransformerConv layer, followed by linear transformation and batch normalization to stabilize training. Batch normalization normalizes inputs to speed up convergence and improve model performance. Multiple TransformerConv layers, each followed by transformation and batch normalization, process the data further. At predefined intervals, TopKPooling layers are applied to reduce the graph size while retaining key structural features. In `forward` function, The global representations from different stages are concatenated using GMP and GAP pooling techniques to capture hierarchical information.

Lastly, `train` is defined to start the learning process of the GNNs. It processes the oversampled train data in batches, computes predictions, evaluates loss, and updates its parameters through optimization techniques. Using `train`, GNNs learns patterns effectively and improves performance over multiple iterations.



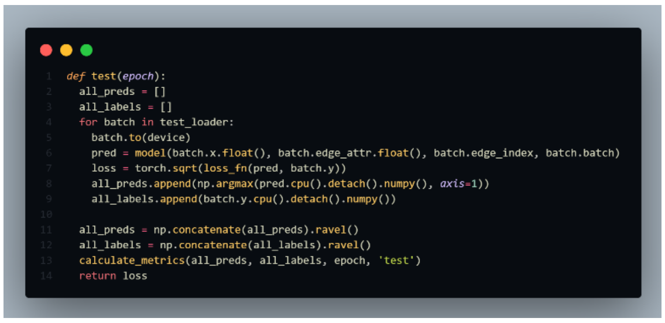
```python
def train(epoch):
  #Enumerate over the data
  all_preds = []
  all_labels = []
  for _, batch in enumerate(tqdm(train_loader)):
    #Recommend to use a GPU
    batch.to(device)
    #Reset Gradients
    optimizer.zero_grad()
    #Passing the node features and the connection info
    pred = model(batch.x.float(), batch.edge_attr.float(), batch.edge_index, batch.batch)
    #Calculating loss and gradient
    loss = torch.sqrt(loss_fn(pred, batch.y))
    loss.backward()
    #Update using the gradients
    optimizer.step()

    all_preds.append(np.argmax(pred.cpu().detach().numpy(), axis=1))
    all_labels.append(batch.y.cpu().detach().numpy())

  all_preds = np.concatenate(all_preds).ravel()
  all_labels = np.concatenate(all_labels).ravel()
  calculate_metrics(all_preds, all_labels, epoch, 'train')
  return loss
```

Fig. 23. GNNs Training Algorithm in Python.

Both versions will be compared by their performance in graph-level classification tasks, providing insights into their strengths and limitations when handling complex graph structures.

D. Evaluation

The test and evaluation phase is designed to assess the model's performance using metrics such as the confusion matrix, precision, recall, and F1-score. It evaluates the model's ability to generalize to unseen data and provides insights into its classification accuracy and reliability.



```python
def test(epoch):
  all_preds = []
  all_labels = []
  for batch in test_loader:
    batch.to(device)
    pred = model(batch.x.float(), batch.edge_attr.float(), batch.edge_index, batch.batch)
    loss = torch.sqrt(loss_fn(pred, batch.y))
    all_preds.append(np.argmax(pred.cpu().detach().numpy(), axis=1))
    all_labels.append(batch.y.cpu().detach().numpy())

  all_preds = np.concatenate(all_preds).ravel()
  all_labels = np.concatenate(all_labels).ravel()
  calculate_metrics(all_preds, all_labels, epoch, 'test')
  return loss
```

Fig. 24. GNNs Test Algorithm in Python.

By running the code in Fig. 24., the model generates predictions and is evaluated using various performance metrics, as shown below.

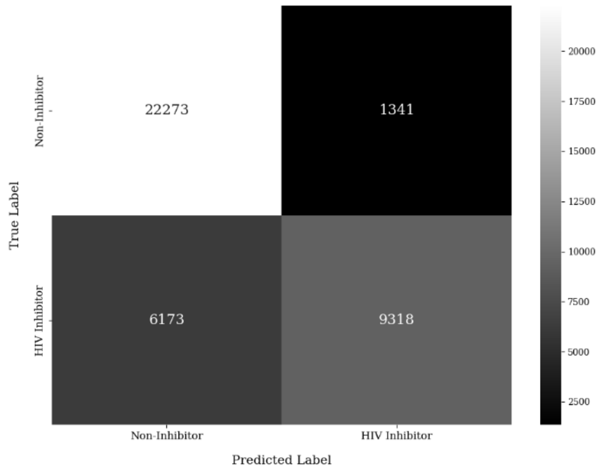


Fig. 25. Convolutional Layers GNNs Confusion Matrix

Fig. 25 illustrates the confusion matrix for the convolutional layers of GNNs, providing insights into the model's potential application in HIV medicines. The high number of correctly identified HIV inhibitors suggests the model could effectively assist in screening compounds for antiviral activity. However, the misclassification of some inhibitors as non-inhibitors highlights the need for further refinement to reduce false negatives, which is critical in drug discovery to avoid overlooking potentially effective treatments.

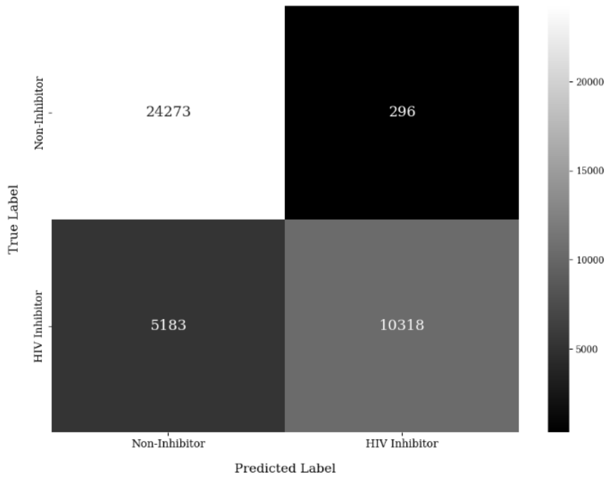


Fig. 26. TransformConv GNNs Confusion Matrix

Fig. 26., showing the confusion matrix for TransformConv GNNs, demonstrates improved performance compared to Fig. 25., which represents the Convolutional Layers GNNs. In Fig. 26., the number of correctly classified non-inhibitors increased from 22,273 to 24,273, and false positives dropped significantly from 1,341 to 296. Similarly, true positives for HIV inhibitors increased from 9,318 to 10,318, while false negatives decreased from 6,173 to 5,183.

By those confusion matrices, the recall, precision, and F1-score were calculated to provide a more detailed evaluation of the model's performance, as shown in Table IV.

TABLE IV
EVALUATION METRIC RESULTS

	Precision	**Recall**	**F1 Score**
Convolutional Layers GNNs	0.872	0.625	0.720
TransformConv GNNs	0.972 ↑ 0.10	0.665 ↑ 0.04	0.790 ↑ 0.07

From a health perspective, this improvement is crucial as it reduces the risk of misclassifying potential HIV inhibitors as non-inhibitors, thereby minimizing false negatives that could lead to missed treatment opportunities. It also lowers false positives, which can prevent unnecessary follow-up investigations on ineffective compounds. These enhancements suggest that the TransformConv GNNs model provides more reliable predictions, making it a better candidate for drug discovery applications, particularly in identifying effective HIV inhibitors.

IV. CONCLUSION

This paper demonstrates the successful implementation of Predictive Graph Neural Networks (GNNs) for identifying HIV inhibitors by leveraging weighted graphs of molecular structures. By representing molecules as graph-structured data, the proposed method effectively captures molecular features and relationships, enabling accurate predictions of inhibitory properties. The model's performance, evaluated through metrics such as precision, recall, F1-score, and confusion matrices, highlights its reliability and robustness in distinguishing between HIV inhibitors and non-inhibitors.

The application of GNNs in molecular analysis not only streamlines drug discovery processes but also provides an innovative approach to studying chemical compounds with biological relevance. The integration of weighted graphs enhances the model's ability to interpret molecular interactions, contributing to more accurate classification and predictive performance. This approach offers a valuable tool for accelerating HIV medicines, minimizing false predictions, and supporting more targeted treatment development.

Along this line, future research may explore more advanced GNN architectures to enrich molecular feature extraction and more reliable predictions. Additionally, expanding the model to analyze more complex molecular structures or drug combinations could further improve predictive accuracy, ensuring broader applicability in healthcare research.

V. APPENDIX

The methods and experiments presented in this paper are implemented in the following GitHub repository: https://github.com/zirachw/Algeo-GNNs

Further explanations of the implementation in this paper are available in the following YouTube video: https://linktr.ee/Zirach

VI. ACKNOWLEDGMENT

The author expresses heartfelt gratitude to God Almighty for His blessings and grace throughout the writing process, allowing the smooth completion of this paper. The author would also like to sincerely thank Ir. Rila Mandala, M.Eng., Ph.D., the IF1220 Discrete Mathematics K02 course lecturer, for his invaluable guidance and knowledge that significantly contributed to completing this work.

Additionally, the author is deeply grateful to their parents for their moral and spiritual support as the source of inspiration and motivation during the writing journey. Finally, the author wishes to thank the friends in the CIPHER cohort for their mutual support during lectures and for learning together throughout the semester.

May the Almighty God reward all the kindness that has been bestowed. May this paper serve as a valuable contribution to the academic community and be well-received.

REFERENCES

[1] HIV.gov. *What Are HIV and AIDS?*, Jan. 13, 2023. Accessed: Jan. 4, 2025. [Online] https://www.hiv.gov/hiv-basics/overview/about-hiv-and-aids /what-are-hiv-and-aids
[2] National Geographic. Accessed: Jan. 4, 2025. [Photo] https://blog.education.nationalgeographic.org/2018/11/29/world_aids_da y_2010/
[3] R. J. B. Escartin. "Success rates and adherence to antiretroviral therapy among treatment-naïve patients in Davao City, Philippines: A ten-year retrospective cohort study". *Dialogues Health*. Sep., 2024. doi: 10.1016/j.dialog.2024.100195.
[4] R. Munir. (2024). *Graf (Update 2024).* Accessed: Jan. 4, 2025. [Photo] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf
[5] Hyperskill. Accessed: Jan. 4, 2025. [Photo] https://hyperskill.org/learn/step/5645
[6] Datacamp. "A Comprehensive Introduction to Graph Neural Networks (GNNs)" Accessed: Jan. 4, 2025. [Online]. Available: https://www.datacamp.com/tutorial/comprehensive-introductiongraph-neural-networks-gnns-tutorial
[7] I. Schaider, E. Frankel, J. Lee. Accessed: Jan. 4, 2025. [Photo] https://medium.com/@schaider/graph-level-neural-networks-for-predicting-social-network-polarization-4916d9589ecc
[8] ResearchGate. Accessed: Jan. 5, 2025. [Photo] https://www.researchgate.net/figure/Examples-of-graph-analysis-tasks-in-three-levels-A-Node-level-the-prediction-of_fig5_353554598
[9] HIVinfo. *GLOSSARY of HIV/AIDS-Related Terms ed. 9.* USA, 2021
[10] D. Weininger. *SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules.* USA. Jun. 1987.
[11] L. Vollmers. Accessed: Jan. 4, 2025. [Onliine] https://luis-vollmers.medium.com/tutorial-to-smiles-and-canonical-smiles-explained-with-examples-fbc8a46ca29f
[12] MoleculeNet. Accessed: Jan. 1, 2025. [Dataset] https://deepchemdata.s3-us-west-1.amazonaws.com/datasets/HIV.csv
[13] B. Tang. "A self-attention based message passing neural network for predicting molecular lipophilicity and aqueous solubility". *J Cheminform,* vol. 12, no. 1, Feb. 2020. Accessed: Jan 1, 2025. doi : https://doi.org/10.1186/s13321-020-0414-z

STATEMENT

In this statement, I declare that this paper I have written is my own work, not a duplication or translation of someone else's paper, and is not plagiarized.

Bandung, 8 January 2025

[illegible]

Razi Rachman Widyadhana
13523004