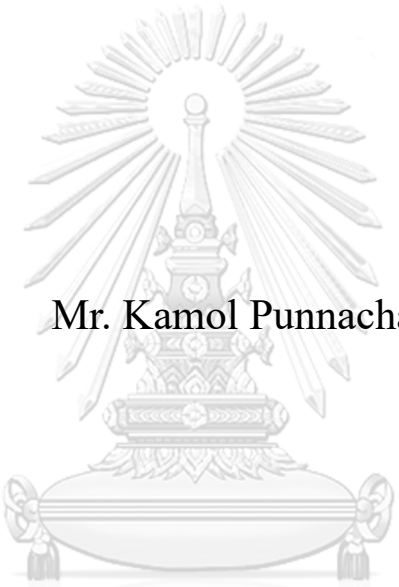
Leverage Graph Neural Network for Molecular Properties Prediction

Mr. Kamol Punnachaiya



A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering
Department of Computer Engineering
FACULTY OF ENGINEERING
Chulalongkorn University
Academic Year 2022

การปรับปรุงเครือข่ายชนิดกราฟเพื่อทำนายคุณสมบัติของสารประกอบทางโมเลกุล

นายกมล ปุณณชัยยะ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2565

Thesis Title	Leverage Graph Neural Network for Molecular Properties Prediction
By	Mr. Kamol Punnachaiya
Field of Study	Computer Engineering
Thesis Advisor	Associate Professor DUANGDAO WICHADAKUL, Ph.D.
Thesis Co Advisor	Associate Professor PEERAPON VATEEKUL, Ph.D.

Accepted by the FACULTY OF ENGINEERING, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Engineering

..... Dean of the FACULTY OF ENGINEERING
(Professor SUPOT TEACHAVORASINSKUN, D.Eng., Ph.D.)

THESIS COMMITTEE

..... Chairman
(Assistant Professor PITTIPOL KANTAVAT, Ph.D.)
..... Thesis Advisor
(Associate Professor DUANGDAO WICHADAKUL, Ph.D.)
..... Thesis Co-Advisor
(Associate Professor PEERAPON VATEEKUL, Ph.D.)
..... External Examiner
(Thanapat Kangkachit, Ph.D.)

กมล ปุณณชัยยะ : การปรับปรุงเครือข่ายชนิดกราฟเพื่อทำนายคุณสมบัติของสารประกอบทางโมเลกุล. (Leverage Graph Neural Network for Molecular Properties Prediction) อ.ที่ปรึกษาหลัก : รศ.ดวงดาว วิชาดากุล, อ.ที่ปรึกษาร่วม : รศ. ดร.พีรพล เวทีกูล

ในยุคของเทคโนโลยีการเรียนรู้เชิงลึก (deep learning) ที่แสดงให้เห็นถึงศักยภาพที่สำคัญในการลดต้นทุนและส่งเสริมการพัฒนาทางการแพทย์อย่างรวดเร็ว การทำนายคุณสมบัติของโมเลกุลเป็นงานหนึ่งที่ได้รับความนิยมและใช้ประโยชน์จากความสามารถของเทคโนโลยีการเรียนรู้เชิงลึก วิทยานิพนธ์ฉบับนี้นำเสนอแบบจำลองกราฟซึ่งรวมโมดูลที่เรียนรู้ชุดข้อมูลเดียวกันจากหลากหลายรูปแบบ (multimodal Graph Neural Network) และใช้ข้อมูลทอพอโลยีที่ได้รับจากกราฟโมเลกุลผ่านแบบจำลองกราฟที่ใช้เป็นเส้นฐาน วิทยานิพนธ์นี้เพิ่มประสิทธิภาพของแบบจำลอง CMPNN ที่ใช้เป็นเส้นฐาน โดยสำรวจวิธีการต่างๆ ที่ยังไม่ได้นำมาใช้ วิธีการเหล่านี้รวมถึงการรวมโมดูลเข้าด้วยกันกับแบบจำลองกราฟ เช่น โมดูล LSTM สองทิศทาง ที่สามารถประมวลผลลำดับของตัวอักษรในรูปแบบ SMILES หรือโมดูลทำสังวัตนาการของกราฟด้วยสเปกตรัม (spectral graph convolution) นอกจากนี้ยังเพิ่มกลไกการเรียนรู้โดยรวมความใส่ใจด้วยตนเอง (self-attention) เข้าในแบบจำลอง CMPNN โดยใช้วิธีการคำนวณตัวเลขอัลฟา (alpha coefficient method) จาก GATConv ผลการทดลองแบบจำลองกราฟที่นำเสนอซึ่งรวมโมดูลที่เรียนรู้ชุดข้อมูลเดียวกันจากหลากหลายรูปแบบ มีประสิทธิภาพโดยรวมดีกว่าแบบจำลองที่ใช้เป็นเส้นฐานในการทำนายคุณสมบัติโมเลกุล จาก 7 ใน 8 ชุดข้อมูลจากโมเลกุลเน็ท ซึ่งประกอบด้วย 5 ชุดข้อมูลในการจำแนกหมวดหมู่ และสามชุดข้อมูลในการทำนายค่า ผลการวิจัยนี้เปิดโอกาสในด้านต่างๆ ในสาขาเคมี โดยเฉพาะอย่างยิ่งในงานค้นพบยา

สาขาวิชา	วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต .....
ปีการศึกษา	2565	ลายมือชื่อ อ.ที่ปรึกษาหลัก .....
		ลายมือชื่อ อ.ที่ปรึกษาร่วม .....

6470133921 : MAJOR COMPUTER ENGINEERING
KEYWOR
D:

Kamol Punnachaiya : Leverage Graph Neural Network for Molecular Properties Prediction. Advisor: Assoc. Prof. DUANGDAO WICHADAKUL, Ph.D. Co-advisor: Assoc. Prof. PEERAPON VATEEKUL, Ph.D.

During the age of deep learning technologies, which have exhibited significant potential in reducing costs and expediting medical development, predicting molecular properties has become a prevalent task that capitalizes on the capabilities of deep learning. This thesis proposed a multimodal Graph Neural Network (GNN) model that utilizes the topology information obtained from molecular graphs through a baseline GNN, facilitating precise property predictions. The thesis improves the baseline CMPNN model by exploring various methods to address potential missing gaps. These methods include incorporating the multimodal module, such as a Bidirectional LSTM module capable of processing text sequences in SMILES format or a spectral graph convolution module. Moreover, self-attention integration into the CMPNN model was implemented using the alpha coefficient method from GATConv. The experimental results show that the proposed multimodal GNN models performed better than the baseline model for predicting molecular properties in seven out of eight datasets from MoleculeNet, including five classification and three regression tasks. These findings show the potential of this methodology across various domains within the field of chemistry, with particular relevance to drug discovery.

Field of Study:	Computer Engineering	Student's Signature .....
Academic Year:	2022	Advisor's Signature .....
		Co-advisor's Signature .....

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my esteemed advisors, Associate Professor Dr. Duangdao Wichadakul and Associate Professor Dr. Peerapon Vateekul. Their unwavering support and encouragement have been instrumental in my academic journey. Whenever I encountered difficulties or experienced a lack of confidence, I could always rely on their attentive ears and genuine guidance.

I am profoundly thankful to Assistant Professor Dr. Pittipol Kantavat and Dr. Thanapat Kangkachit for graciously serving as members of my thesis committee and providing invaluable comments and suggestions.

Kamol Punnachaiya

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

Chapter 1
Introduction

1.1 Motivation

Previously, to develop a new drug for treatment in the medical field, scientists needed to discover each compound's properties in the laboratories experimentally. A manual approach to such a large combination of compounds and ways to experiment will hinder the speed of finding new drugs, not even including the amount of funds and resources to sink into these experiments.

In recent times, the pandemic of COVID-19 has destroyed many lives, and it is incredibly fast to evolve and improve in their resistance to vaccines; However, the amount of time and resources needed to research new drugs/vaccines for fighting these diseases are enormous. Hence, the disease remains spreading infection an expansive and fast-paced manner.

With the help of recent blooming technology, Machine-Learning and Deep-Learning play a considerable role in substantially decreasing the time and resources needed to develop a new drug/vaccine to fight constantly mutated diseases. We can now develop machine-learning models to precisely predict molecule properties by feeding molecular representation vectors into the model to train and learn to predict an unseen molecule's properties, bypassing the experimental steps in the process. Their primary use is to help screen for potential compounds that can be used for making new drugs. Hence, its contributions to the medical field are invaluable.

This thesis aims to discover new ways for an improved precise way to predict molecular-level properties by using their molecule's graph structure and features, which the results can be further used to assist in developing new drugs.

1.2 Objectives

Try to improve graph neural network deep learning model architecture by integrating new modern methods into the existing molecular properties prediction model and obtain better performance metrics than the state-of-the-art methods.

1.3 Scope

The models will be evaluated on graph benchmarks from MoleculeNet datasets [1] including BBBP, ClinTox, SIDER, Tox21, HIV, ESOL, FreeSolv, and Lipophilicity.

1.4 Expected Results

Obtain a deep learning model that achieves higher performance metrics than the baseline model.

1.5 Research Plan

1. Research state-of-arts methods for graph neural networks.
2. Research state-of-arts methods for graph neural networks focusing on drug property predictions.
3. Investigate if any method can integrate with current work and obtain better performance.
4. Prepare the datasets.
5. Implement new methods into the model and evaluate the model performance.
6. Evaluate and analyze the results.
7. Summarize and publish the research results.
8. Defend the thesis.

1.6 Publication

K. Punnachaiya, P. Vateekul and D. Wichadakul, "Multimodal Modules and Self-Attention for Graph Neural Network Molecular Properties Prediction Model" April 21st-23rd, *2023 11th International Conference on Bioinformatics and Computational Biology (ICBCB)*, Hangzhou, China, 2023.

Chapter 2

Related Theories and Literature Review

2.1. Related Theories

Before studying previous works, it would be beneficial to investigate some essential background and overview for easier understanding. Therefore, in this section, it will consist of the following subsections:

1. Hierarchy of graph neural network (GNN) terminology
2. K-hop neighbors
3. Message Passings' categorization specifically in this thesis.

2.1.1 Hierarchy of graph neural network (GNN) terminology

2.1.1.1 Category: Recurrent and Convolutional GNNs

The main difference between recurrent GNNs and convolutional GNNs is how they propagate information through the graph structure.

1. **Recurrent GNNs**: recurrent GNNs propagate information through the graph by sequentially updating the hidden state of each node based on its neighboring nodes' hidden states. At each time step, the hidden state of a node is updated by aggregating and transforming the hidden states of its neighboring nodes. This process is repeated for a fixed number of iterations or until convergence. Recurrent GNNs are often used for tasks that require capturing complex temporal dependencies or for graphs with variable-sized neighborhoods.

2. **Convolutional GNNs**: convolutional GNNs, also known as spatial GNNs, propagate information through the graph by applying convolution operations to the node features and their local neighborhoods. Instead of updating nodes sequentially, convolutional GNNs operate on the entire graph simultaneously (please note that updating nodes on the entire graph does not necessarily needs the whole graph structure to be precomputed; instead, it can use their direct neighbors). Each node's updated feature is computed by aggregating information from its local neighborhood using a convolutional operation, similar to how convolutional neural networks operate on images. Convolutional GNNs are suitable for tasks that can benefit from local neighborhood information.

Sharing weights across multiple message passings or convolution operations does not count towards being categorized as Recurrent GNNs. To be Recurrent GNNs there needs to be RNNs within and utilize temporal dependencies. The node update stage needs to explicitly depend on the previous step rather than just purely on the current step.

Figure 1 shows two approaches of graph neural networks (GNNs): recurrent graph neural network (Rec-GNN) and convolutional graph neural network (Conv-GNN) [2].

The main difference between these two variants is that Rec-GNNs have connections between perceptron that can form cycles, or their outputs are re-inputted into the model for multiple iterations, which allows them to reuse information from previous hidden layers and retain memories from the past. In contrast, the weights in each layer of Conv-GNNs are not shared, so the parameters of each layer are not affected by the previous layer.

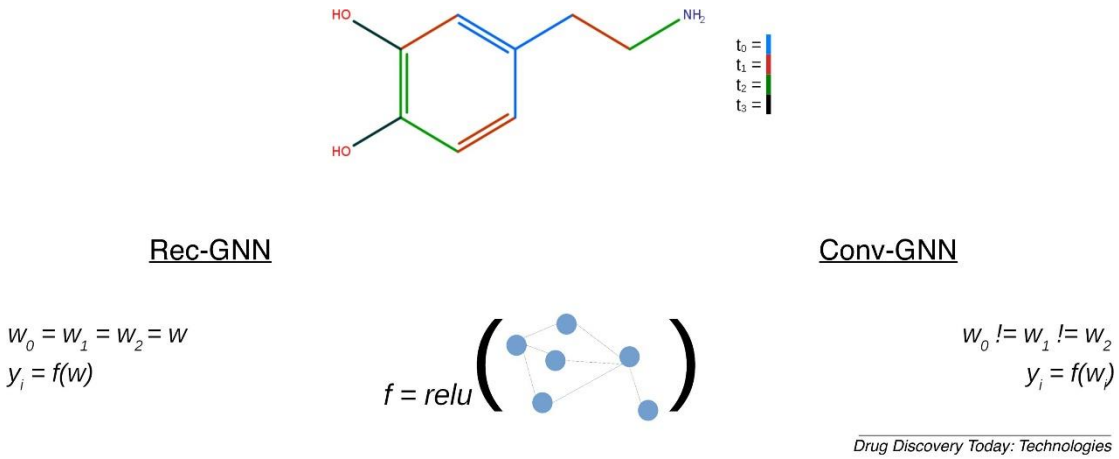


Figure 1: Recurrent GNN vs Convolutional GNN [2]

Sharing weights can reduce the number of parameters and being computationally more efficient than non-shared weight layers. This efficiency means the layer gains a form of dynamic memory that can store information from previous timesteps, but this memory can be overwritten or updated by new information, which is not the case in separate weight layers.

The node vector update formula for Rec-GNNs is given below in Equation 1 and Conv-GNNs in Equation 2 [2].

$$h_u^{(t)} = \sum_{v \in N(u)} \rho(M_w([x_u, x_{u,v}^e, x_v, h_u^{(t-1)}])) \quad (1)$$

$$H^t = \rho(AH^{t-1}W^{(t)}) \quad (2)$$

2.1.1.2 Variant: Spectral and Spatial

Spectral GNNs

Spectral GNNs are inspired by spectral graph theory and leverage the eigenvalues and eigenvectors of the graph's Laplacian matrix. Spectral GNNs use the graph's spectral domain, which is analogous to the frequency domain in signal processing. Spectral GNNs transform the graph data into a spectral representation by performing eigenvalue decomposition of the Laplacian matrix.

In spectral GNNs, graph convolution is performed by filtering the graph's spectral representation by applying filters in the spectral domain to capture different frequency components of the graph. By convolving the graph's spectral features with these filters, spectral GNNs can capture global structural patterns and relationships across the graph. Spectral GNNs are particularly effective for tasks that require capturing long-range dependencies in the graph.

They are effective for tasks requiring understanding the overall graph structure and spectral characteristics. The only downside being computationally expensive and may struggle with large graphs due to the eigenvalue decomposition process.

Spatial GNNs

Spatial GNNs, on the other hand, operate in the spatial domain of the graph, which is similar to the time domain in signal processing. Spatial GNNs propagate information through the nodes and edges of the graph in a localized manner. They typically aggregate information from neighboring nodes and update node representations based on this local neighborhood information.

In spatial GNNs, graph convolution is performed by aggregating features from neighboring nodes and combining them with the current node's features. This allows spatial GNNs to capture local connectivity patterns and propagate information across the graph. Spatial GNNs, e.g., graph convolutional networks (GCNs), and message passing neural networks (MPNNs), are well-suited for tasks that require capturing local interactions and dependencies within the graph. They are also more scalable and suitable for larger graphs.

2.1.1.3 Approach: GCNs and MPNNs

An approach refers to different methods used within the broader category variant, in this thesis, we will study two types of spatial GNNs approaches: GCNs and MPNNs.
GCN has two functions, which are aggregation and readout, while MPNN has three functions, which are message function (equivalent to aggregation in GCN approach), update function, and readout function.

The difference between the two approaches is the additional update function in MPNN. However, either the aggregate function in GCN or the message function in MPNN involves multiplying the feature matrix with an adjacency matrix.

Below are the most straightforward functions for each approach as stated in [2], with the following symbols definitions t - timestep, ρ - non-linearity function, w - weight, M - message, O - readout, N(u) - neighbors of vertex u, G - graph, v - vertex, h - hidden node representation, ˆy - graph representation.

GCN:

$$h_u^{(t)} = \sum_{v \in N(u)} \rho(M_{w_i}^{(t-1)} h_v^{(t-1)})$$

$$\hat{y} = O_{w_j}(\sum_{v \in G} h_v^{(t)})$$

MPNN:

$$m_u^{(t)} = \sum_{v \in N(u)} \rho(M_{w_i}^{(t-1)} h_v^{(t-1)})$$

$$h_u^{(t)} = U_{w_k}^{(t-1)}([h_u^{(t-1)}, m_u^{(t-1)}])$$

$$\hat{y} = O_{w_j}(\sum_{v \in G} h_v^{(t)})$$

2.1.2 K-Hop neighbors

Figure 2 depicts the message passing framework, in which messages from k-hop distant neighbors are passed to the target node through multiple iterative/depth update steps in the Message Passing Neural Network [2].

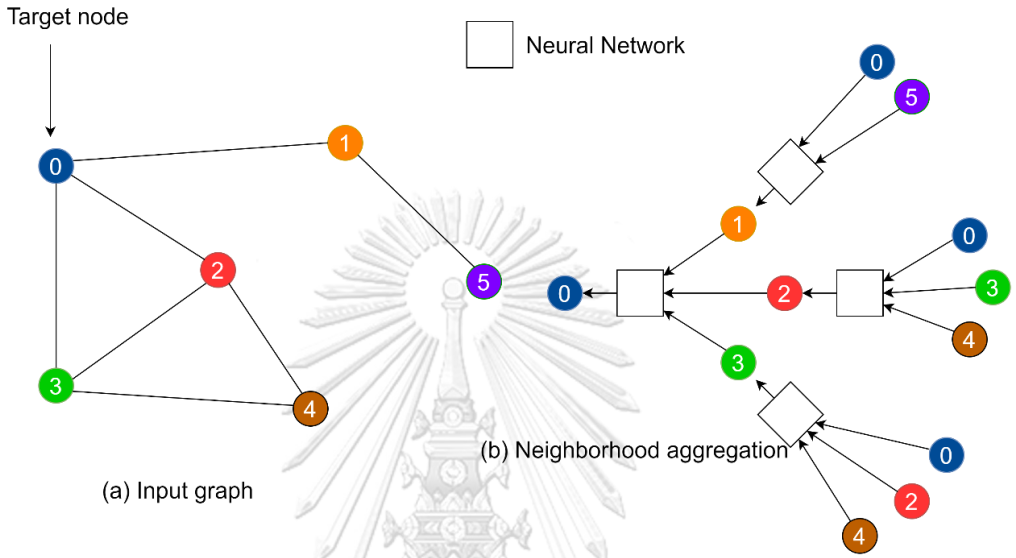


Figure 2: Visualization of K-Hop layers, based on [3]

2.1.3 Message Passings' categorization

In this thesis specifically, we studied/summarized four variants of message passing neural networks from the following papers: [4] [2] [5] and [6]. All fall under the spatial convolutional graph neural networks and utilize the MPNN approach.

2.1.3.1 Atom message passing

In Atom Message Passing, messages are situated directly within the graph's nodes. The process starts with aggregating information from neighbors. Initial messages are oriented in atoms, only containing atom features.

The neighboring messages, atoms, and bonds must be indexed and selected separately. The selected neighboring atom/bond message types are then concatenated at a hidden dimension and aggregated as the total incoming message using standard summation. These aggregated messages are then passed directly to their neighboring atoms via the connectivity matrix in the next iteration.

2.1.3.2 Bond message passing

The messages are located on the edges of the graph, each edge containing two hidden states (messages), one for each direction.

The initial message, which is oriented/stored at bonds, is combined with the atom features that the bonds originate from. This results in the message being different in opposite directions. As the bond message is already combined with atom features, only one step for selecting incoming neighboring messages is required.

The reverse bond message (either the initial outgoing bond message or the outgoing bond message from the previous step) is subtracted from the updated node representations. The subtracted message becomes the new neighboring bond message when passing the message to neighboring atoms through connecting edges in the next iteration.

Having the atom and bond messages updated concurrently in each iteration of the message passing positively affects the model's performance.

2.1.3.3 Undirected graph message passing

In an undirected graph, edges do not have a specific direction, meaning both directions of an edge are equivalent and constant.

For bond message passing, where bi-directional messages from each direction containing different information are possible, both messages are averaged to eliminate the direction distinction. On the other hand, atom message passing does not change the message as there is no direction in the first place. This algorithm provides base results.

2.1.3.4 Directed graph message passing

This type of message passing is only applicable in bond-oriented message passing because of the presence of two directions in a single bond.

In this approach, the direction of the message is significant and can differ (bi-directional: outgoing and incoming). The selected neighboring messages for updating the node representation are only from the incoming messages in the direction of the target atom.

This algorithm enables the model to avoid passing redundant messages to its neighbors.

Summary

In short, bond message passing (i.e., directed) has been improved from atom message passing (i.e., undirected) stores messages within the graph's edge and updates both node and edges simultaneously and utilizes both directions of an edge while atom message passing only store messages within its node and update only nodes within 1 iteration.

Previously, machine learning in the medical/chemical fields was limited to traditional machine learning methods. However, with the advancements in deep neural networks, graph neural networks have become a popular choice for molecular properties prediction, as presented below.

2.2 Spatial Conv-GNNs with MPNN approach

2.2.1 MPNN(Gilmer, J., et al.)

MPNN is an approach to graph neural networks introduced by Gilmer et al. in the paper "Neural Message Passing for Quantum Chemistry" [4]. The model focuses on creating high-quality node embeddings using undirected graphs and atom message passing. The authors emphasized the importance of making the model invariant to graph isomorphism, as a graph has no inherent order to nodes. Furthermore, they noted that its performance might suffer if the model is given or injected with low-quality node positional encodings as additional features.

2.2.2 DMPNN

This work [5] introduces a new method to message passing by using directed graphs and subtracting overlapping bond messages when passing messages to neighbors. This subtraction separates the relevance of the two-direction bond messages in the same bond, optimizing the message passing process and avoiding passing messages to unnecessary loops. The improved quality of the final node embeddings directly leads to better results after pooling the node embeddings into graph embeddings.

The MolGraph class developed by the authors in [5] and used in DMPNN is crucial as it serves as a molecule feature extraction method. It allows for a bi-directional message passing through the edge, and its success has been utilized in several other studies on molecular property predictions. The authors also created a framework, Chemprop [5], based on this MolGraph class.

Figure 3 and Figure 4 illustrate the most simplistic overview of edge message passing. Figure 3 shows the first step to aggregate all incoming green messages to the yellow atom. In the second step, Figure 4, the blue aggregated incoming message is then passed along the outgoing bonds subtracting out the

initial red outgoing messages to not pass overlapping messages to the neighboring atoms.

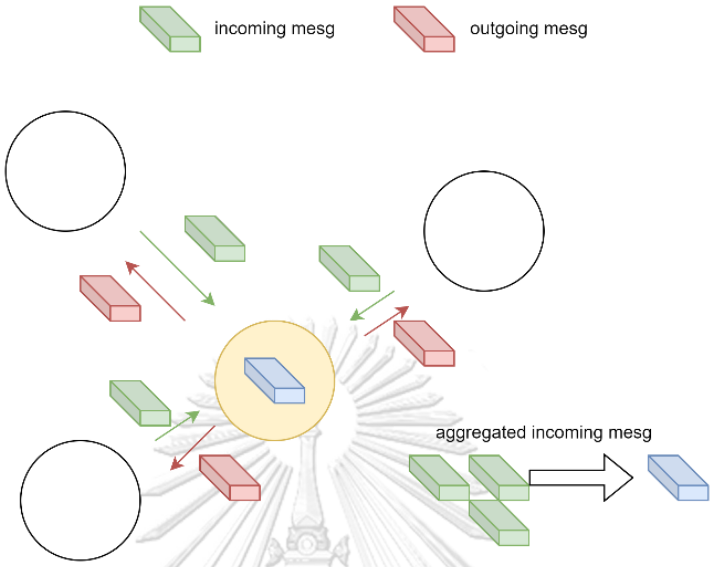


Figure 3: Aggregation of green incoming messages

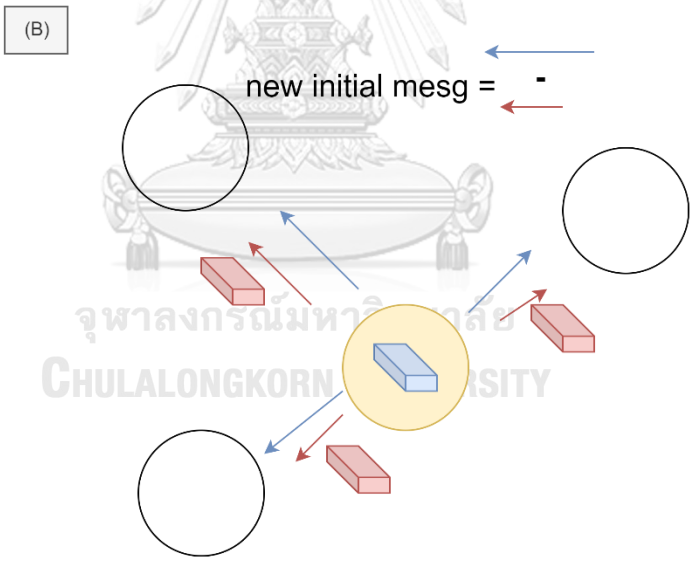


Figure 4: Subtraction out red overlapping messages from the outgoing message

Figure 5 shows the original figure describing the directed message passing from DMPNN, which will also be explained for another perspective of understanding.

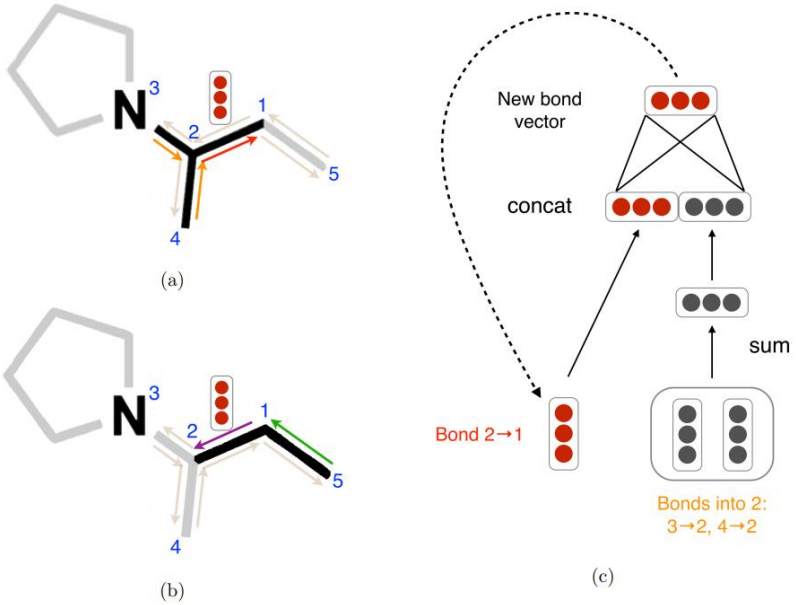


Figure 5: Edge message passing from DMPNN [5]

Consider atom number 2 (blue number), which has two incoming orange bonds and one incoming grey bond in the opposite direction to the outgoing red bond. After aggregation (note that this is not the updated node representation), it creates a total incoming neighboring message. When sending off the aggregated and updated message by using the outgoing bonds indices of atom number 2, the red outgoing message will get subtracted by the initial outgoing message (initial message 2->1) so as not to create a redundantly updated initial incoming message for atom number 1 in the next iteration.

Figure 6 and *Figure 7* demonstrate the code of edge message passing, where *Figure 6* represents the aggregation of all initial incoming bond messages. a_message becomes the total incoming messages. *Figure 7* shows the initialization of reverse bond messages selection for every bond in the matrix, which means the outgoing bond index the incoming bond and vice versa. This gave outgoing bond messages as a result.

```
nei_a_message = index_select_ND(message, a2b)  #
a_message = nei_a_message.sum(dim=1)  # num_atom
```

Figure 6: DMPNN's code: message selection

a_message[b2a] is then expanded by indexing each aggregated message (i.e., a_message) with an outgoing bond index (i.e., b2a[.]) This gives the current outgoing messages, which needs to be subtracted by initial outgoing messages. Subtractions are performed to avoid sending the overlapping of the

outgoing message, which will then be used as incoming messages for neighboring atoms.

```
rev_message = message[b2revb]  # num_bo
message = a_message[b2a] - rev_message
```

Figure 7: DMPNN's code: subtraction of overlapping message when passing messages to its neighbors

Figure 8 illustrates how DMPNN updates a node's state, which consists of two steps: update node representation and update edge message.

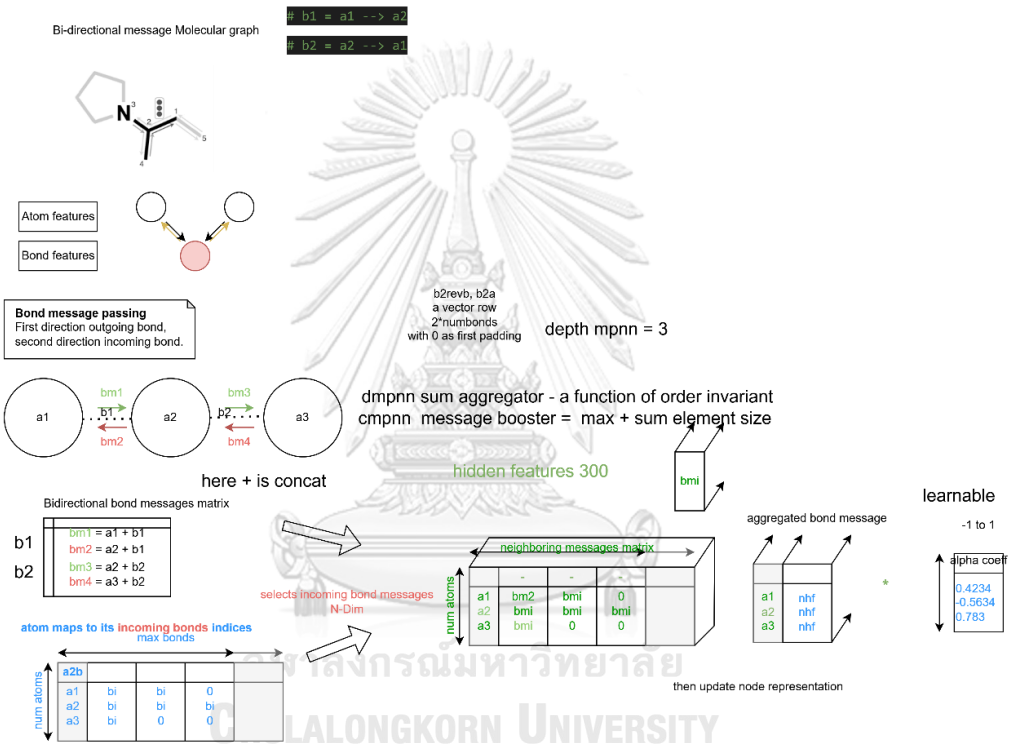


Figure 8: DMPNN's node update

DMPNN's node update

In the code implementation of bond-message passing, it first creates a bidirectional bond messages matrix that consists of each message being the featured bond itself concatenated with the atom features that the bond came from. The two bond directions are the outgoing and the incoming directions. As seen in *Figure 8*, the green and red bond messages of the same bond connectivity align directly below one another.

The adjacent indices are obligatory to get neighboring messages for example, the more commonly used edge adjacency list or adjacency matrix.

In this case, DMPNN introduced a new type of graph connectivity matrix called N-Dimensional neighbor mappings. The blue matrix on the left is

the bond index mappings for each atom incorporated with zero paddings at the end of each row to the maximum number of current bonds neighboring an atom in the present molecular graph.

With access to both the bidirectional bond messages and bond mappings for each atom, the model can select the neighboring message to get the next green matrix containing the neighboring messages of each atom.

At this stage, the model has done the aggregation between columns to create aggregated neighboring messages for each atom. This aggregated/reduced matrix is then used to update the representations of the nodes/atoms in the MPNN using an update function. Figure 9 illustrates the neighboring message update.

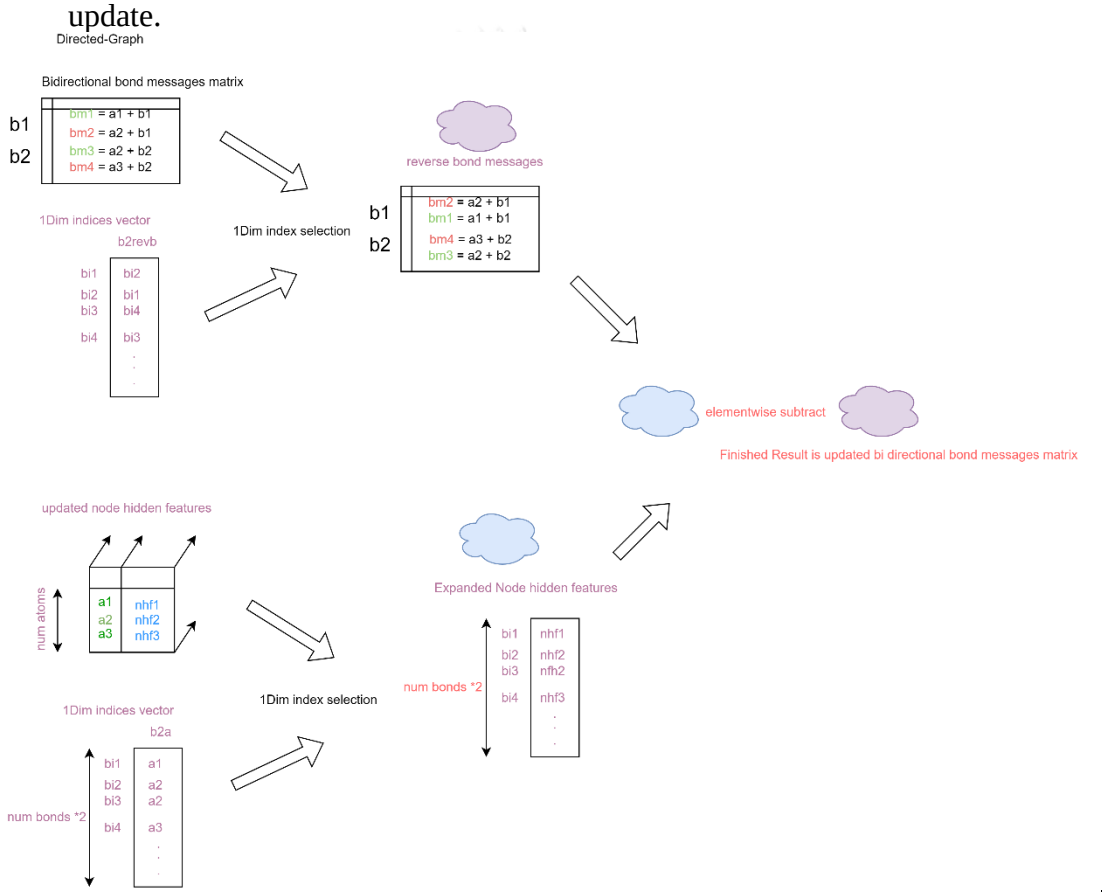


Figure 9: DMPNN's edge message update

DMPNN's edge message update

In this step, the bidirectional bond messages matrix will be reused together with the bond to reverse bond mappings. After performing normal indexing, the model creates a matrix of the same shape as the bidirectional bond messages; however, containing messages with reverse direction instead, swapping each pair of rows as in the upper middle of Figure 9.

Next, the model indexes the updated node representation matrix with the bond to atom mappings to expand the number of rows and matches atom

representations with their corresponding incoming bonds. Now, the augmented matrix can be directly subtracted out of the reverse bond message from the earlier swapping matrix. The resulting matrix will then be used as the new bidirectional bond message matrix in the next iterations.

2.2.3 CMPNN

The authors of CMPNN [6] suggested a new way to improve the existing DMPNN model by introducing a message booster N-D aggregation. This message booster works by sum aggregating node representations and element-wise multiplying with max aggregation of node representations. This message booster (Figure 10) helps preserve more information from the local sub-structures after aggregation.

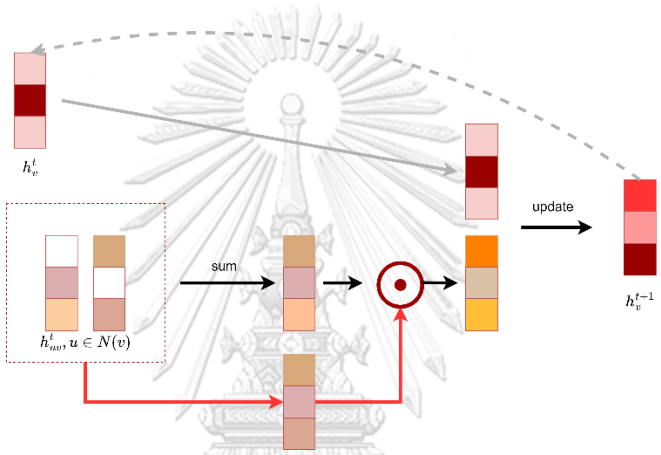


Figure 10: Message booster based on [6]

Additionally, the authors modified gated recurrent units (GRU) to handle batch mol graphs from the data loader successfully.

They used GRU for its update and reset gates which helps the model learn to keep or discard certain long-range information from the past, in this case, generalization of graph expressiveness from node representations updating with their neighboring messages. The Bi-directional GRU reads a concatenation of aggregated node representations, aggregated neighboring messages, and initial atom embeddings at the hidden dimension.

The GRU takes the above-mentioned concatenated node representations as input and initializes the GRU cell hidden weight matrix to be the same as the concatenated node matrix.

Lastly, the input messages are each added with a learnable bias and then activated with ReLU activation function. The row-wise matrix zero paddings to the maximum number of atom rows in the current molecular batch are needed for consistency in the shape of sequence messages length. The padding is obligatory to feed into the GRU successfully.

Finally, the output of GRU resulted in a more fine-grained hidden node representation. This final representation of node embeddings is then processed

with mean pooling to produce a better graph representation than typical pooling, which only uses an order-invariant function without the GRU refinement.

After extensive studies in the code implementation of CMPNN architecture, we found some questionable implementations. First, the update node stage did not include the Communicative function. Second, the BatchGRU proposed receives concatenated node representations as input but initializes the GRU cell hidden representation shaped as the concatenated node matrix instead of a more logical aggregated neighboring message. After some revision, the cause of the second whole aggregation was to be more aware and optimize all of them at once with more learnable capability than using just an aggregated neighboring messages matrix as input. We may further investigate those mentioned above and whether the changed implementation can improve model performance.

2.2.4 GROVER

This work [7] proposes a new architecture for molecular graph encoding, using multi-head attention blocks (i.e., auxiliary module) inspired by the transformer in Natural Language Processing (NLP) but without node positional encoding.

The molecular graph is encoded into high-dimensional vectors using a message passing network encoder (MPNEncoder), which is a type of graph neural network (GNN), in place of the linear modules in the original transformer architecture (as seen in Figure 11 pink-highlighted). In this work, the authors use DMPNN as the MPNEncoder.

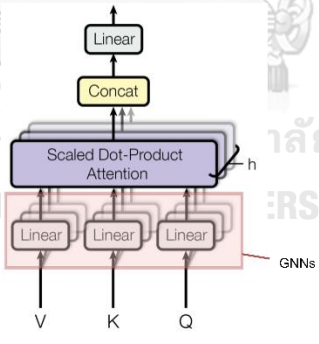


Figure 11: Multi-head attention from transformer [8]

The logits or prediction vectors generated from each MPNEncoder are used as Value, Key, and Query, and each is later fed into a linear module and calculated the same way as a multi-headed attention function in the transformer model. The formula of scaled dot product attention Equation 3 and Equation 4.

attention scores = $softmax(\frac{QK}{\sqrt{d_k}})$, attention scores (i.e., alpha coefficients) (3)

predictions = V * attention scores, predictions (i,e., logits) (4)

This multi-headed attention mechanism lets the model see the data's hidden features from different angles.

2.3 Spatial Conv-GNNs with GCN approach

2.3.1 GCNConv

The graph convolutional network (GCN) [9] proposed by Kipf and Welling is a type of spatial GNN that operates on the graph structure using the normalized graph Laplacian matrix. GCN employs the Chebyshev polynomial approximation of the spectral filter to simplify the convolution operation.

In GCN, the convolutional operation is defined as a propagation rule aggregating and combining information from a node's local neighborhood. This operation can be formulated in terms of the graph Laplacian, which represents the connectivity structure of the graph. The graph Laplacian matrix captures the relationships between nodes and their neighboring nodes.

To simplify the spectral filter, GCN utilizes the Chebyshev polynomial approximation. By approximating the spectral filter with a Chebyshev polynomial of order K=2, GCN can effectively capture information from up to 2-hop neighbors of a node. This approximation allows GCN to capture localized information and learn hierarchical representations of the graph structure.

To ensure stability and avoid issues related to scaling, GCN normalizes the graph Laplacian matrix before applying the Chebyshev polynomial approximation. This normalization step involves dividing the graph Laplacian by the maximum eigenvalue, which scales the matrix and ensures that the convolution operation remains stable during training.

2.3.2 GATConv

The Graph Attention Network (GAT) [10] proposed a self-attention mechanism in addition to GCN [9].
The steps for implementing self-attention mechanisms are as follows

1. Initialization: Given an input graph with nodes and their corresponding feature representations, GATConv initializes learnable parameters, including weight matrices, for each node.

2. Attention coefficients: For each node, GATConv calculates attention coefficients by computing a compatibility score calculation that involves concatenating the node's feature vector $\vec{h}_i$ and the neighboring node's feature vector $\vec{h}_j$ using the concatenation operator (||) and then applying a linear transformation using the weight matrix W. The concatenation is passed through

the activation function (LeakyReLU) to obtain the compatibility score e_{ij} (Equation 5).

$$e_{ij} = LeakyReLU(a^T(W\vec{h}_i \,||\, W\vec{h}_j)); \quad (5)$$

where a^T is the transpose of the learnable attention vector.

3. Attention weights: The compatibility scores are then passed through a SoftMax function, which normalizes the scores across all neighboring nodes, ensuring that the weights sum up to 1. This step creates attention weights that reflect the relative importance of the neighboring nodes for the central node.
The attention weight for each edge (i, j) is given by Equation 6 as follows.

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ik})} \quad (6)$$

where N*i* represents the set of neighboring nodes of node i.

4. Aggregation: The attention weights are used to compute a weighted sum of the neighboring node features. The features of the neighboring nodes are multiplied by their corresponding attention weights and summed together to obtain a weighted feature representation.
The aggregated feature representation for node i is given by Equation 7 below.

$$\acute{h}_i = \sigma(\sum_{j\in N_i} a_{ij} \cdot Wh_j) \quad (7)$$

where σ is an activation function, such as ReLU or LeakyReLU.

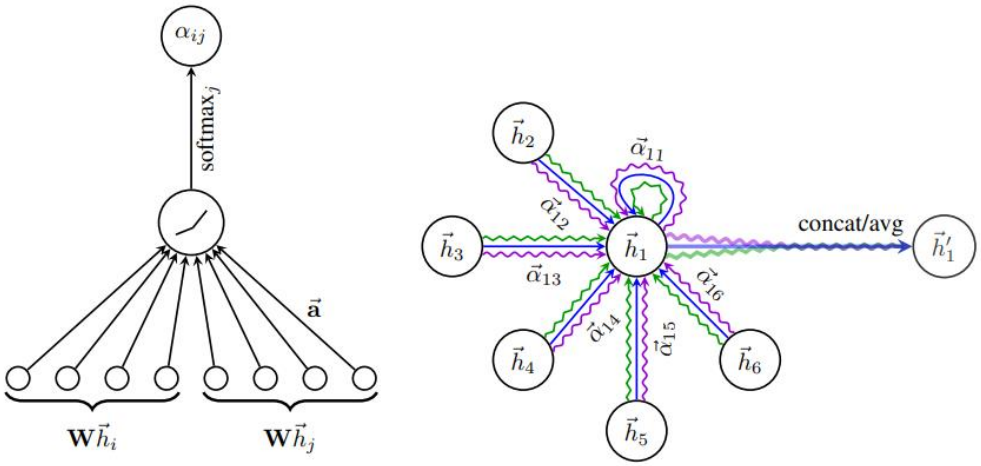


Figure 12: GATConv alpha calculation [10]

Comparison between GCN and GAT

Equation 8 shows GCN, fixed and normalized alpha by degree, while GAT is dynamic learnable, as shown in Equation 9.

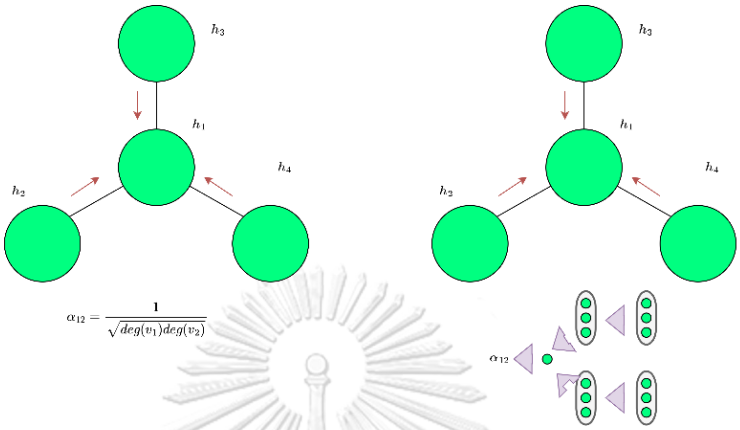


Figure 13: Simplified distinction between GCN and GAT based on DSG IITR [11].

$$\alpha_{ij} = \frac{1}{\sqrt{deg(v_i)deg(v_j)}} \tag{8}$$

$$\alpha_{ij} = \frac{\exp\left(LeakyReLU(\vec{a}^T[W\vec{h}_i||W\vec{h}_j])\right)}{\sum_{k\in N_i}\exp\left(LeakyReLU(\vec{a}^T[W\vec{h}_i||W\vec{h}_k])\right)} \tag{9}$$

2.3.3 GINConv

GINConv [12] introduced the following algorithm.

***Algorithm*: *WL* − *algorithm* (*Weisfeiler* & *Lehman*, 1968)**

Input: $Initialnodecoloring\left({h_1}^{(0)}, {h_2}^{(0)}, \dots, {h_N}^{(0)}\right)$

Output: $Final\ node\ coloring\ \left({h_1}^{(T)}, {h_2}^{(T)}, \dots, {h_N}^{(T)}\right)$

$t \leftarrow 0$;

repeat

$for\ v_i \in V$ ***do***

$h_i^{(t+1)} \leftarrow hash(\sum_{j \in N_i} {h_j}^t)$

$t \leftarrow t + 1$;

until stable node coloring is reached;

The algorithm is based on the Weisfeiler-Lehman (WL) isomorphism test, also known as a graph coloring algorithm. It handles the issue of graph isomorphism, where multiple graphs have the same number of nodes, degree, and connectivity but appear differently tangled.

$$h_v^{(k)} = MLP^{(k)}((1 + \epsilon^{(k)} \cdot h_v^{(k-1)} + \sum_{u\in N(v)} h_u^{(k-1)})) \tag{10}$$

Equation 10, the proposed learnable epsilon at the target node feature representation allows the model to adjust the relative importance of the central node's representation compared to its neighbors and MLP to learn a complex injective function to map node representations to unique high dimensional vectors.

The study found that sum aggregation is a practical order-invariant function. Most importantly, it proposed an injective function (in this case, MLP, a 1 to 1 function) which operates on multisets of node features and maps them to a unique high dimensional vector. The resulting high-dimensional space allows for improved mapping of isomorphic graphs [12].

2.3.4 Graph Transformer

The authors [7] of this paper added node positional encoding as an additional input feature in the model, which is derived from eigenvector decompositions and the WL algorithm. However, this approach did not help generalize the model to unseen graphs and instead reduced the validation loss.

The paper also stated that injecting a node's positional encodings without careful consideration could lead to a worse performance due to the added noise, as it violates the inductive bias of permutation equivariance that graph neural networks (GNNs) are designed for [13].

Summary

DMPNN and CMPNN enhance the accuracy of graph regression and classification by removing unneeded loops in the message passing process with edge-oriented, directed message passing, leading to more refined node representations.

Other baseline Conv-GNNs do not use bi-directional message passing. Hence the importance of edge-message passing makes CMPNN a good candidate for improvement. Moreover, GROVER does not modify the message passing paradigm directly. Thus, CMPNN is selected as the baseline to be explored in this thesis.

Table 1 shows the semi-supervised classification accuracy from GATConv on three datasets - Citeseer, Cora, and Pubmed. The results show that GAT outperforms GCN [10] with an accuracy of 83.0% ± 0.7%, 72.5% ± 0.7%, and 79.0% ± 0.3%, respectively. The performance of GATConv and the ability to be used on any graph with self-attention mechanisms make it an appealing option for baseline improvement. Hence, this thesis also tried implementing self-attention mechanisms into the selected baseline.

Table 1: Semi-supervised classification accuracy as reported from GATConv

	Citeseer	Cora	Pubmed
GCN	81.5%	70.3%	79.0%
GAT	83.0 ± 0.7%	72.5 ± 0.7%	79.0 ± 0.3%

Besides using CMPNN as the baseline and deciding to incorporate self-attention mechanisms into CMPNN, we revisited the origin of GCN, which was supposed to be from the spectral domain, for additional ideas of baseline improvement.

2.4 Spectral graph convolution

True spectral graph convolutional neural networks (GCNNs) are based on the convolution theorem. The convolution theorem states that the Fourier transform of the convolution of two functions equals the element-wise product of their Fourier transforms.

In spectral GCNNs, the graph signal (node features) and the graph filter (spectral weights) are transformed into the spectral domain using the graph Fourier transform. The graph Fourier transform diagonalizes the graph Laplacian matrix and allows us to operate on the graph signals in the frequency (spectral) domain.

Once the graph signal and filter are in the spectral domain, the convolution theorem can be applied. The convolution operation is performed by element-wise multiplication of the Fourier transforms of the graph signal and the graph filter. This allows for efficient computation and propagation of information through the graph.

However, this approach must apply the convolution filter to the entire graph in the spectral domain, making it challenging to scale to larger graphs and requiring pre-computed graph connectivity.

As a result, this method is limited to transductive tasks; the model only learns on a pre-computed and fixed graph structure, meaning that the graph structure is known before the learning process starts and remains constant. Unlike in inductive tasks, the model must be able to learn arbitrary and unknown graph structures, meaning that the graph structure is not known beforehand and can change during the learning process. The steps to perform spectral graph convolution are as follows:

1. Obtain the graph's Laplacian matrix (or adjacency matrix). This matrix represents the connectivity of the graph.
2. Compute the eigenvalues and eigenvectors of the Laplacian matrix. The eigenvalues represent the frequencies or spectral components of the graph.
3. Select a set of eigenvalues (frequency centers) to be used as the convolution supports. These eigenvalues will determine the frequency range over which spectral convolution will operate.

4. Design a filter function that assigns weights to the eigenvalues based on their proximity to the selected frequency centers. This filter function should determine the contribution of each eigenvalue to the spectral convolution.
5. Multiply the Laplacian matrix by the filter function to obtain the convolution supports in the spectral domain. This will result in a modified Laplacian matrix where the entries correspond to the weights assigned to each eigenvalue.
6. Use the modified Laplacian matrix in the spectral convolution operation to propagate node features across the graph.

This paper [14] mainly aims for graph isomorphism tasks. The authors did their work by designing a graph spectral filter. Also, they studied matrix language to break the limit of MPNN for the model to gain the ability to distinguish cospectral graphs.

In the frequency/spectral domain, we can transform graphs into a signal for spectral graph convolution as a type of graph signal filtering. We must first convert the convolution kernel to its Fourier form (spectral domain) to calculate graph convolution in the spectral domain. Then after the graph signals have been filtered, the results are converted back to the original time domain.

From spectral graph theory, signals are defined on a graph, where the graph structure defines the convolution operation. Spectral graph convolutions are often performed in the graph Fourier domain, obtained from the eigenvectors of the graph Laplacian matrix (L). Eigen decomposition of the graph's Laplacian matrix generates U and λ as eigenvalues and eigenvectors, respectively.

$L = U\Lambda U^T$, Λ is a diagonal matrix with eigenvalues (λ) of L; $\Lambda = diag(\lambda)$

The steps for applying the spectral filter/frequency response function to the graph signals are as follows. First, apply Fourier transform to the graph signal (features) x. Then apply with a spectral filter $F(\lambda)$ (i.e., $\hat{g}(\Lambda)$). Lastly, reverse Fourier transforms back to the original domain. This results in the final filtered graph signal x. These steps are given below.

1. $U^T x$ 2. $\hat{g}(\Lambda)U^T x$ 3. $U\hat{g}(\Lambda)U^T x$ 4. $x_{filtered} = U\hat{g}(\Lambda)U^T x$

There are many ways to design a spectral filter, as mentioned by [14], where the authors [14] proposed to design a filter function as described in the following section.

2.4.1 Spectral designed convolution support/filter

The spectral convolution support is related to the spatial domain power of the graph's adjacency matrix; instead, the spectral uses eigenvalues and eigenvectors, both determine the extent of the interaction between nodes. The convolution support (C'), as shown below, determines the set of nodes

considered in the computation of the output signal for a given node and is used to control the smoothness of the output signal.

$$C' = Udiag(F(\lambda))U^T, where\ F(\lambda)\ is\ the\ desired\ spectral\ filter\ function$$

$$x\_filtered = Udiag(F(\lambda))U^T x$$

The filtered graph signal x applied by frequency response function $F(\lambda)$ resulted from using eigenvalues of Laplacian matrix as $input$ $(i.e., \lambda)$, where $F(\lambda)$ had to be learnt by back propagation ($\Phi_s(\lambda)$ is an alias of $F(\lambda)$).

The authors [13] proposed using spectral graph convolution to break through the limit of MPNN, which mostly has a ceiling of 1-WL performance. In the paper, frequency response functions were designed to be sensitive to each spectrum of the graph signal, as in the following formula.

$\Phi_s(\lambda) = -b(\lambda - f_s)^2, f_s \in [\lambda_{min}, \lambda_{max}]$, where b is the bandwidth of the bandpass filter (i.e., spectral filter). Hence, the full spectral convolution supports C' can fully be written as

$$C' = Udiag\big(\Phi_s(\lambda)\big)U^T$$

This convolution support could be seen as extracted edge features. These features are then multiplied with a sparse connectivity matrix mask, which will propagate the features through the graph's structure defined as $M * C' = Udiag\big(\Phi_s(\lambda)\big)U^T$,where M is a sparse connectivity matrix and can be a power of adjacency matrix, which encodes the number of walks possible from $node_i$ to $node_j$ in a matrix entry ij. The paper used these filtered signals x as edges features and multiplied with a learnable weight to be learnt through backpropagation. This sums up how the graph spectral convolution works.

The authors reported the results being theoretically as powerful as 3-WL or 2-FWL. Still, they stated that in most real-world datasets with a lot of node and edge features, 1-WL would suffice. After that, with $C = M * C'$ as mentioned above, the authors proposed a convolution in Equation 11.

$$H^{l+1} = \sigma(\textstyle\sum_s C^{(s)} H^{(s)} W^{(l,s)}) | mlp_5(H^l \odot \mathrm{mlp}_6(H^{(l)})) \qquad (10)$$

2.4.2 GNNML3 (MATLANG 3 Layer)

Matrix Language (MATLANG), The author [14] studied this theory and found that trace and element-wise multiplication matrix operations are needed to break the limit of 1-WL MPNN to 3-WL. Each Lx is a set of operations at x-WL levels where . is a dot product, $diag$ is diagonalize, tr is trace, $\odot$ elementwise,

1 is column vector full of 1 and f is element-wise custom function operating on scalars or vectors.

$$L_1 = \{., transpose, 1, diag\}$$
$$L_2 = \{., transpose, 1, diag, tr\}$$
$$L_3 = \{., transpose, 1, diag, tr, \odot\}$$
$$L^+ = L \cup \{+, \times, f\}\ where, L \in \{L_1, L_2, L_3\}$$

$tr(A) = \sum_{i=1}^{n} a_{ii}$, Trace matrix operation is the sum of diagonal elements.

By defining convolution supports in the spectral domain to filter graph signals, we can propagate features using spectral structure features of the graph. Spectral GNNs main advantages were to provide rich global information and dependencies of a huge complex graph. This thesis also incorporated GNNML3 as a multimodal module.

Chapter 3

Methods

3.1 Data preparation and Preprocessing

MoleculeNet [15] is a widely used benchmark for evaluating the performance of deep learning models on molecular data. It provides data in a CSV format, with each row representing a molecule in the form of SMILES strings and its target properties.

The characteristics of each dataset used in this paper are detailed in *Table 2*. These datasets consist of 2D molecular graphs represented only in SMILES format and do not include 3D molecular information. Therefore, they differ from other datasets in MoleculeNet, such as quantum mechanics datasets, which contain 3D molecular information.

Table 2: MoleculeNet dataset details [15]

Dataset	Type	#Tasks	#Compunds
ESOL	Regression	1	1128
Tox21	Classification	12	7831
BBBP	Classification	1	2039
Lipophilicity	Regression	1	4200
ClinTox	Classification	2	1478
SIDER	Classification	2	1427
HIV	Classification	1	41127
FreeSolv	Regression	1	642

We experimented on the following dataset for regression tasks:

1. ESOL - Water solubility data
2. FreeSolv - Calculated hydration free energy of molcules in water
3. Lipophilicity - Octanol/water distribution coefficient

We experimented on the following dataset for classification tasks:

1. BBBP - Binary labels of blood-brain barrier penetration
2. ClinTox – Binary Clinical trials results for toxicity
3. SIDER - Marketed drugs and adverse drug reactions (ADR)
4. Tox21 - Qualitative toxicity measurements
5. HIV - Abilities to inhibit HIV replication

SMILES (Simplified Molecular Input Line Entry System) [16] is a string format system that uses a string sequence to represent the molecule.

The rules of the notation can include Branches, Rings, Charged Atoms, Atom chirality, Aromatic, stereochemistry, and explicit Hydrogens. These rules allow the

SMILES sequence to contain all the information to reconstruct back to its 2D molecular graph.

3.2 Preprocessing stage

The data preprocessing was done using the RDKit [17] and Chemprop libraries [5]. We used RDKit to process SMILES strings and MolGraph from Chemprop to extract molecular features, including one-hot encoded and normalized raw values, as listed in *Table 3*.

Table 3: Chemprop's features

Atom Features	Type	Description
	Atomic number	Range 0~99.
	Degree	Range 0~5. The atoms directly connected bonds and whether the Hydrogen is explicit in the molecular graph.
	Formal charge	Range -2~2
	Chiral tag	Range 0~3. Chirality refers to the property of a non-superimposable molecule on its mirror image. In other words, a chiral molecule exists in two forms that are mirror images of each other, and these two forms cannot be superimposed on one another. The chiral tag of an atom specifies its configuration in space.
	Number of Hydrogens	Range 0~4. Number of hydrogens in direct neighbors.
	Hybridization type	5 types. Ex. sp, sp2, sp3, sp3d, sp3d2
	Aromatic	A compound that exhibits aromaticity. Aromaticity is a property of some organic compounds characterized by a ring of atoms that is unusually stable and has a high degree of delocalization of the electrons in the ring.
	Atomic mass	* 0.01 for scaling
Bond Features	Type	Description
	Bond type	Range0~4 Ex. zero padding, sing, double, triple, aromatic.
	Conjugate	whether the bond is conjugated.
	In ring	whether the bond is within a closure ring of the molecule.
	Stereo	Range 0~6. Spatial arrangement of

		atoms around a double bond or a triple bond.
Custom edges matrices	Type	Description
	a2a	Mappings of atom to their neighboring atoms. 2D Matrix's shape (num atoms, max num atoms).
	a2b	Mappings of atom to their incoming bonds indices. 2D Matrix's shape (num atoms, max bonds).
	b2a	Mappings of bonds to atom that they came from. 1D Matrix's shape (2*num bonds).
Note for custom edges matrices	Mappings do not contain the elements to be mapped, only the indices being mapped to. However, the sequence all corresponds to the SMILES string sequence. For the bonds, pair-wise combination indices bonds between atoms are created.	

Note that we can convert molecular representation in RDKit (RDKit.Mol) to PyTorch Geometric [18] or PyG graph object; however, PyG objects only support edge-list or sparse adjacency matrix in COO format, which may not allow for bi-directional message passing. Nevertheless, the significance of bi-directional message passing for updating node embeddings has been highlighted in the literature, so this thesis decided to use the Chemprop library for featurization as the basis.

With the data and features extracted and adopting CMPNN as the baseline, we proposed improving CMPNN with the following entities.

- Attention mechanisms
- Multimodal
    1. Text features
    2. Spectral features

3.3 Attention mechanisms and Alpha coefficients computations

We proposed combining GATConv and a shared-weight convolution kernel to CMPNN to allow the model to capture better the overall information and relationships in the target matrix. In addition, to avoid inaccuracies, zero padding in the atom-to-bond indices (i.e., a2b) mapping matrix from Chemprop is ignored during the computation of alpha coefficients.

Initially, the linear modules used for projection were not allowed to have an additive bias, as this would result in non-zero values after the attention mechanism was applied. However, after redesigning, we used a mask created from the a2b matrix in conjunction with masked SoftMax to ignore zero indices after kernel multiplication.

This implementation distributes a learnable percentage to each neighboring bond connected to an atom, allowing the model to learn whether it should amplify or reduce each neighboring message before aggregation. We illustrate the matrix operations and details of this self-attention in *Figure 14*.

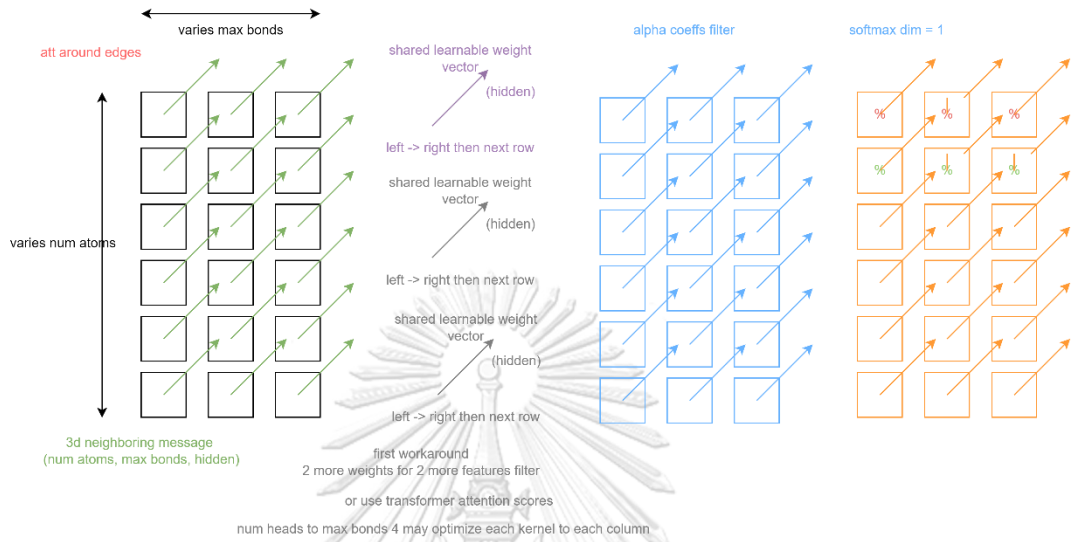


Figure 14: Attention mechanism visualization on neighboring messages

The first green matrix is the N-Dim selected neighboring messages with zero-adding in the column dimension if that atom has a smaller number of bonds than the maximum number of bonds in this batch of molecular graph. The purple vector will multiply every element in the green matrix to create a blue matrix. We considered this blue matrix an attention filter. as in computer vision. The grey vectors can be added and used like the first purple vector. Both the purple and grey vectors will create multiple blue attention filters. These attention filters can then be aggregated to become an attention filter ready for the last step.

In the last step, the aggregated attention filter will be activated using masked SoftMax attention to distribute the attention/focus of the model whether it should pay attention to which corresponding edges while ignoring the zero padding indices for each atom in the molecular graph.

The fully modified algorithm from CMPNN can be displayed in *Algorithm 1*.

Algorithm 1: CMPNN based formula

$h^0\left(e_{v,w}\right) \leftarrow x_{e_{v,w}}, \forall e_{v,w} \in E; h^0(v) \leftarrow x_v, \forall v \in V$
$for\ k = 1 \dots K\ do$
$for\ v\ \in V\ do$
$\left\{h^{k-1}\left(e_{u,v}\right), \forall u \in N(v)\right\} \leftarrow attentionedges(\left\{h^{k-1}\left(e_{u,v}\right), \forall u \in N(v)\right\})$
$m^k(v) \leftarrow AGGR(\{h^{k-1}\left(e_{u,v}\right), \forall u \in N(v)\})$
$h^k(v) \leftarrow communicate(m^k(v), h^{k-1}(v)\ )$
$end\ for$
$for\ e\ \in E\ do$
$m^k(e_{v,w}) \leftarrow h^k(v) - \ h^{k-1}\left(e_{v,w}\right)$
$h^k\left(e_{v,w}\right) \leftarrow \sigma\left(h^0(e_{v,w)} + W.m^k(e_{v,w})\right)$
$end\ for$
$end\ for$
$m(v) \leftarrow AGGR(\{h^K\left(e_{u,v}\right), \forall u \in N(v)\})$
$h(v) \leftarrow communicate(m(v), h^K(v), x(v))$
$z \leftarrow Readout(\{h(v), \forall v \in V\})$

$AGGR\left(\left\{h^{k-1}\left(e_{u,v}\right), \forall u \in N(v)\right\}\right)$
$= sum\left(\left\{h^{k-1}\left(e_{u,v}\right), \forall u \in N(v)\right\}, dim = 1\right)$
$* \max\left(\left\{h^{k-1}\left(e_{u,v}\right), \forall u \in N(v)\right\}, dim = 1\right); where$
$* \ is\ hardamand\ product\ and$ dim 1 $is\ neighboring\ dimension.$

$communicate\ or\ update\left(m^k(v), h^{k-1}(v)\right) = \ h^{k-1}(v) + \ m^k(v); where +$
$is\ elementwise\ addition.$ This function applies for 1 to K-1 iterations.

$communicate_K or\ update_{K\left(m^k(v), h^{k-1}(v), x(v)\right)}$
$= \ GRU\left(concat(m^k(v), h^{k-1}(v), x(v)\ at$ dim 1$)\right); where$ dim 1 $is\ hidden\ features\ dimension.$

$Readout(\{h(v), \forall v \in V\}) = mean(\{h(v), \forall v \in V\}, dim = 0); where$ dim 0 $is\ row.$Averaging node features to graphs feature correspondingly to atom scope from Chemprop.

$alpha = dropout(Softmax(LeakyReLU(W_{attention}.$ (lin($\left\{h^{k-1}\left(e_{u,v}\right), \forall u \in N(v)\right\}$))))
$attentionedges = alpha * \left\{h^{k-1}\left(e_{u,v}\right), \forall u \in N(v)\right\};\ where *\ is\ hardamand\ product.$

3.4 Multimodal/External feature extraction models for MLP classifier

We proposed two multimodal modules that can be trained concurrently with the CMPNN, and the concatenated graph vectors are injected into an MLP classifier, as shown in Figure 15.

The multimodal module box can be either one of the following: a multimodal text module that extracts additional features by transforming each character in the SMILES to a one-hot vector, then feeding it into a Bi-directional LSTM (a popular NLP module) to memorize long-term dependencies from the sequence.

Alternatively, we can use the GNNML3 [14] module, which uses Matrix Language 3 to perform graph signal processing for spectral graph features and is believed to retain more global structure information [2]. The MLP is created by stacking two linear layers with 300 default dimensions and ReLU activation between the layers. The output molecular vector from the GNN (our baseline CMPNN) has 300 hidden features, while the Bi-LSTM has 300*2 features.

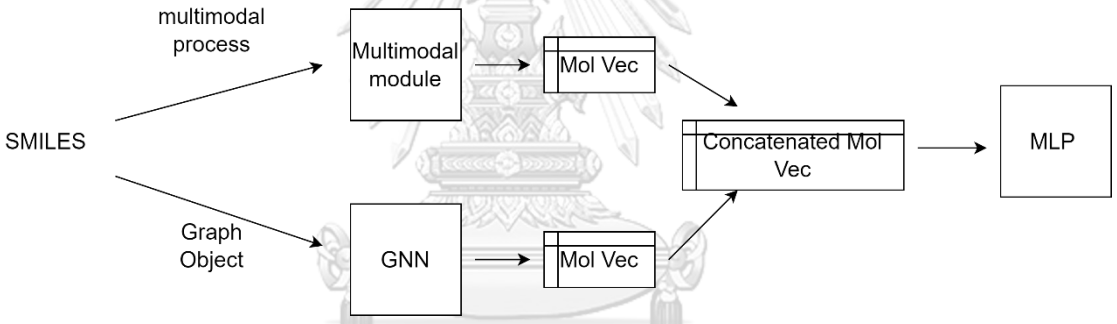


Figure 15: Multimodal GNN architecture

3.4.1 Text Multimodal Module Using Bi-LSTM

3.4.1.1 Raw one-hot encoded text feature matrix for Bi-LSTM

In this experiment, we introduced encoding each character in a SMILES sequence into a matrix. The dimension of each one-hot vector of a character will be according to all possible numbers of characters in ASCII format, which is 128.

As in Figure 16, a 2D square matrix contains a one-hot vector of each character in the SMILES string, and the total number of vectors is equal to the total number of characters in that SMILES string. We used zero padding to make the number of vectors in each square matrix consistent. Each square matrix gets batched together in the 3rd dimension according to the batch size for batching.

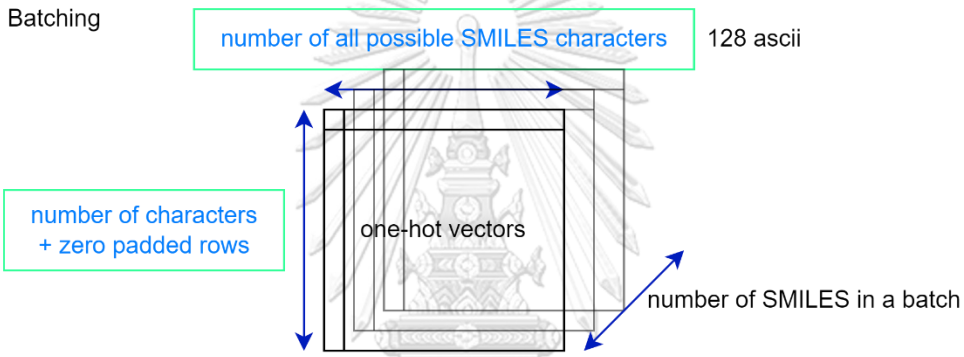


Figure 16: SMILES sequence one-hot encoding to text feature matrix.

The above one-hot setting resulted in the final dimension of (batch size, max SMILES length, max ASCII). These matrices were fed into Bi-LSTM to read and learn the processed text feature matrices. Bi-LSTM helps ensure the model understands the context of a SMILES sequence's forward and backward directions, as there is no canonical character order in the SMILES sequence.

Each LSTM predicts the next character feature vector (i.e., next one-hot character vector) of this SMILES sequence, but after the model learns/optimizes, it can output a molecular vector representing the whole SMILES sequence instead (Figure 17).

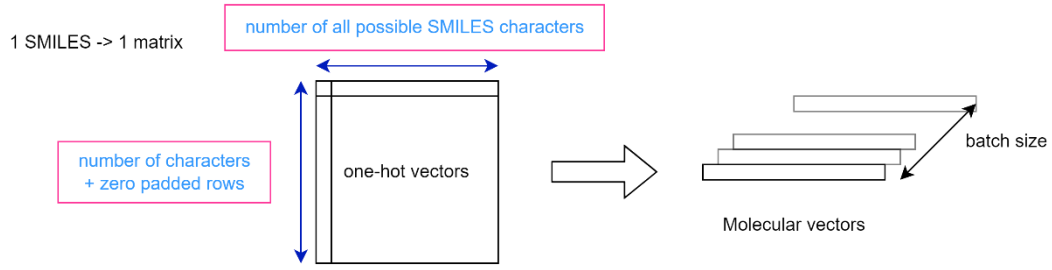


Figure 17: LSTM predictions

The result of both directions of LSTMs' predictions and their final cell states were then concatenated into a batch of molecular vectors. Subsequentially, both outputs from each direction of LSTM were then aggregated using mean operation. These molecule vectors are then further concatenated with other molecular vectors from other modules to feed into MLP for the final prediction (Figure 18).

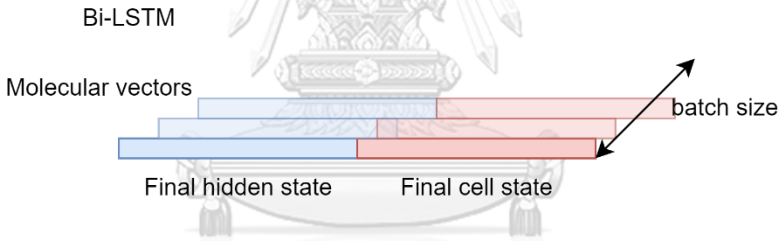


Figure 18: Bi-LSTM concatenation

3.4.1.2 Further improvement on the initial one-hot encoding SMILES using embedding lookup table

We could optimize the text feature matrix before feeding into Bi-LSTM utilizing an embedding layer. First, we made a list containing the ASCII number of each character in the SMILES string and zero-padding to the maximum number of SMILES length (Figure 19).

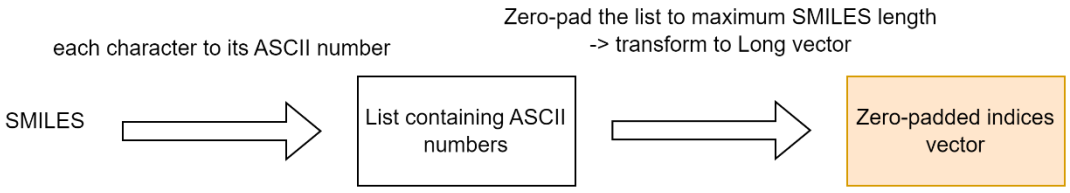


Figure 19: SMILES indices vector

We passed this indices' vector into an embedding lookup table to index and get the corresponding learnable vector from the table for each index (i.e., each ASCII character becomes a learnable word in the embedding layer) (Figure 20).

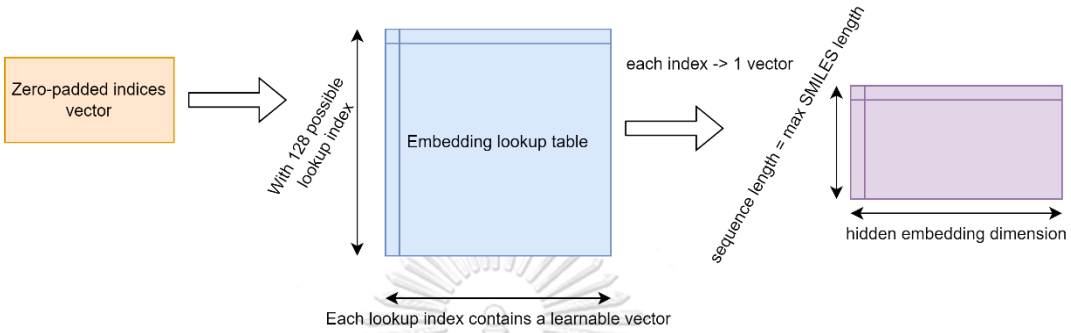


Figure 20: Indices lookup from embedding table

We then batched all resulting 2D matrices for all SMILES in each batch (Figure 21).

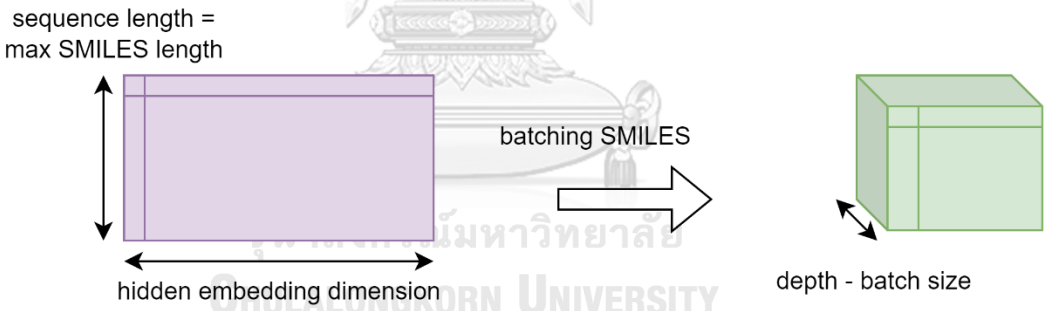


Figure 21: Batching of 2D SMILES matrices

Finally, we fed the batch to the Bi-LSTM, which read each 2D matrix of the batch containing a character-embedded vector in each row. Once the LSTM reads the whole sequence, it will learn the context of the SMILES string. In this case, we concatenated both the final hidden state of Bi-LSTM and the final cell state (Figure 22). Final hidden state, which arguably predicts the next character embedding vector, will eventually work as an aggregator, turning the SMILES string vectors into a representation vector for that molecule (i.e., global pooling). In addition to this, the final cell state contains crucial information about the long-term learning dependencies throughout the training process (Figure 23).

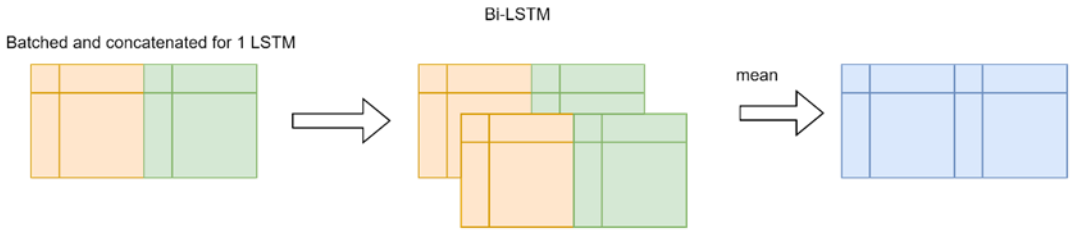


Figure 22: Final concatenation and mean of both forward and backward

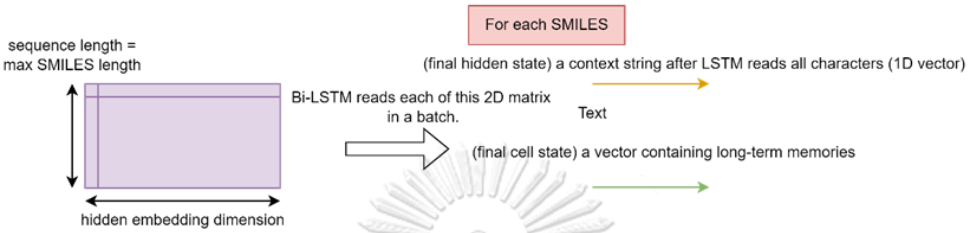


Figure 23: Output of LSTM

All these processes result in a batch of molecular features extracted from Bi-LSTM. The model can transform the SMILES string into a learnable vector by adding this embedding layer.

3.4.2 Spectral features multimodal

We tried directly computing spectral filter for each batch graph while training instead of precomputing a whole graph of the dataset (and arguably impossible since the data in the dataset are not the type that all connected as a single huge graph) and used the unmodified GNNML3 from [14] as a multimodal module.

3.5 Model Architecture and Training Procedure

3.5.1 Model Hyperparameters

The model hyperparameters include k-hop depth, hidden feature sizes, dropout probabilities, and the number of feed-forward linear layers in the last MLP before prediction.

These parameters can be optimized using grid search, but for fair out-of-the-box comparisons with the baseline CMPNN, the unoptimized version of 300 hidden size was used. In addition, we used ReLU as the activation function between layers before feeding into the MLP. The details parameters are listed in *Table* 4 and *Table 5*.

3.5.2 Details of training and Predicting Process

We trained the model using Chemprop's default five-fold cross-validation with the recommended split type from MoleculeNet into three subsets: train, validation, and test, with aspect ratios of 0.8, 0.1, and 0.1.

Stochastic gradient descent was used as the optimizer, with a normalized and scheduled learning rate for 30 epochs and a batch size of 50. To further improve the training process, we also utilized Chemprop's early stopping mechanism, where the best model based on the validation loss was saved and checkpointed for testing on unseen data.

A norm learning rate scheduler with a piecewise linear increase and exponential decay from the "Attention is All You Need" paper was also incorporated [8].

Table 4: Proposed MPNN Layer's parameters

Linear Atom input	133, 300
Linear Bond input	133+14, 300
Depth/K-hop	3
Depth 0 bond linear	300, 300
Depth 1 bond linear	300, 300
BatchGRU	300, 300*2 (bi-directional)
Linear out after bi-directional BatchGRU	600, 300

Table 5: MLP for prediction depth = 1

Linear	300, 300
ReLU	1 activation
Linear	300, number of tasks in dataset

We employed the learning rate scheduler because it can help the model converge faster by adjusting the learning rate to allow the model to make progress early on while avoiding getting stuck in local minima. In addition, the normalization can help ensure that the learning rate is appropriate for the model's current state, improving the training process's stability. It can also potentially improve the final accuracy of the model by fine-tuning the learning rate during the training process. Learning rate scheduler details can be seen in Table 6.

Table 6: Learning rate scheduler

Initial learning rate	1e-4
Maximum learning rate	1e-3
Final learning rate	1e-4
Warmup epochs	2

The model training flow starts by creating and initializing the model's weights with one fixed seed. The model was then trained and tested for five iterations, each iteration with a different dataset split seed. Finally, the five test results were used to calculate the mean and standard deviation (Figure 24).

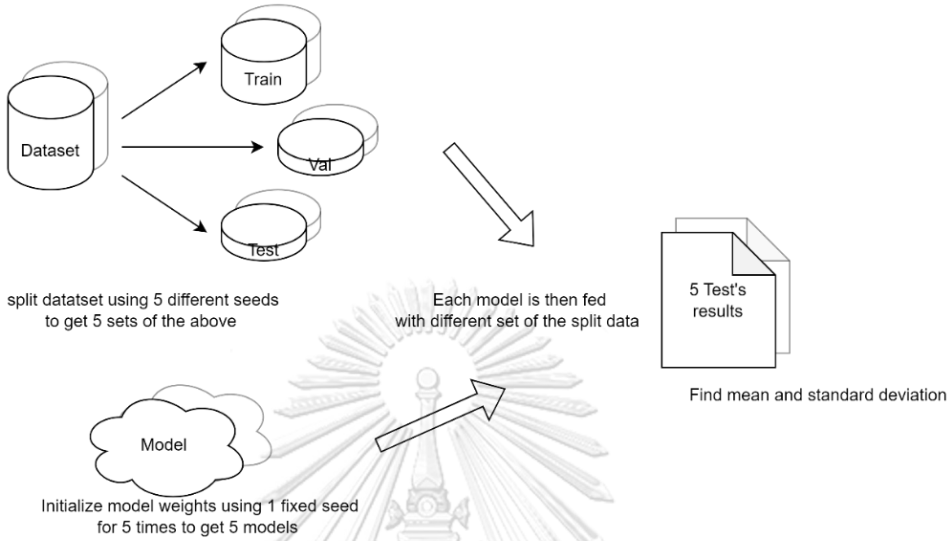


Figure 24: Training flow

3.6 Evaluation Metrics

The molecular properties prediction tasks comprise two categories: graph regression and graph classification tasks. We used the evaluation metrics of each category as follows.

3.6.1 Classification Tasks

We evaluated the model's performance for the classification task using a true positive rate (tpr) and false positive rate (fpr), as shown in Equation 12 and Equation 13,

$$tpr = TP / (TP + FN) \quad (12)$$

$$fpr = FP / (FP + TN) \quad (13)$$

where TP is true positive, TN is true negative, FP is false positive, and FN is false negative. We then plotted the receiver operating characteristic (ROC) curve using tpr as y- and fpr as x-axes and calculated the area under the curve (AUC). The higher, the better.

3.6.2 Regression Tasks

We calculated Root Mean Square Error (RMSE) using Equation 14, where $Y\hat{}$ and Y are the predicted and observed values, respectively. The lower, the better.

$$RMSE = \sqrt{\frac{1}{N}\sum(Y\hat{} - Y)^2} \quad (14)$$

Chapter 4

Results and Discussion

This chapter presents the experimental results of CMPMM incorporated with various modules or techniques for drug property prediction tasks.

4.1 Performance of the Baselines

We describe the naming definitions of each model implementation of the baselines as follows.

- CMPNN: The baseline GNN that took advantage of the disconnected bi-directional edge messages.
- GNNML3: The spectral baseline model utilizing spectral graph convolution with MATLANG3 from [14] with the second power of adjacency (K=2) (direct hop per message passing like normal spatial GNN).

Table 7 shows the baselines' performance implemented in this thesis and evaluated by five-fold cross-validation using recommended split types from MoleculeNet.

Table 7: Five-fold cross-validation results of our implemented baseline models

Five-fold cross valida-tion	**Classification**					**Regression**		
	BBBP	**ClinTox**	**SIDER**	**Tox21**	**HIV**	**ESOL**	**Free-Solv**	**Lipophi-licity**
CMPNN (Base-line)	**0.958077 +/- 0.017582**	**0.915815 +/- 0.015748**	**0.640620 +/- 0.039283**	**0.851490 +/- 0.009426**	**0.808721 +/- 0.020270**	**0.582865 +/- 0.048626**	**0.969931 +/- 0.335350**	**0.583237 +/- 0.021075**
GNNML3 (Spec-tral) (K=2)	0.761271 +/- 0.058912	0.431808 +/- 0.130315	0.558689 +/- 0.016746	0.742669 +/- 0.025536	0.707289 +/- 0.037594	1.364229 +/- 0.156225	3.520960 +/- 0.320290	1.128885 +/- 0.043017

Even though CMPNN was entirely better than the MATLANG3 layer, we keep in mind that the MATLANG3(spectral) was designed to be used in the graph isomorphism task rather than the graph classification/regression tasks like CMPNN. The spectral features and the extracted graph isomorphism information may still be useful for prediction.

Table 8 shows the calculated average molecular graph size and average denseness of each dataset's direct neighboring bonds of an atom. This information may help us analyze the results in the future (Section 4.2).

Table 8: Datasets' average number of atoms and bonds

	Classification					Regression		
	ESOL	Tox21	FreeSolv	BBBP	Lipophi-licity	ClinTox	SIDER	HIV
Avg atoms	13.289	18.574	8.722	24.064	27.04	26.157	33.641	25.510
Avg bonds	0.988	1.009	0.915	1.066	1.089	1.049	1.021	1.072

*Avg atoms: Avg number of heavy atoms per smiles; Avg bonds: Avg number of bonds per heavy atom

4.2 Performance of the CMPNN variants

This section presents the results of CMPNN incorporated with various techniques and multimodal modules. We describe the naming definitions of each CMPNN variant as follows.

- AttnMaskedSM: Self-attention at edges, distributed along the target node's edge with the masked SoftMax function.
- GNNML3: The spectral graph convolution using the second adjacency power (K=2), a spectral multimodal module running concurrently with the standard CMPNN.
- S2SPool: Popular attention pooling operator from Set2Set [19] used for pooling node feature matrix to a molecular graph vector.
- Bi-LSTM: The multimodal module, Bi-directional LSTM, fed with raw one-hot vectors of characters in SMILES string.
- Bi-LSTM+EMB: Bi-LSTM with pre-processing of transforming SMILES string to learnable vector via a lookup table.

Table 9 shows the overall performance of CMPNN variants compared with the baselines, and Table 10 summarizes the number of winning tasks of each CMPNN variant compared with the baseline.

Table 9: Model performance of CMPNN variants compared with the baseline.

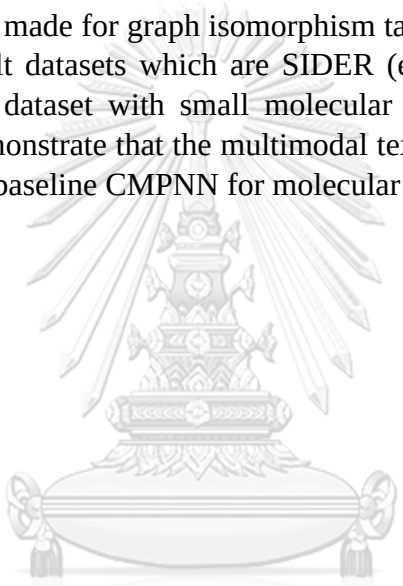
Five-fold cross-validation	**Classification**					**Regression**		
	BBBP	**ClinTox**	**SIDER**	**Tox21**	**HIV**	**ESOL**	**FreeSolv**	**Lipophilicity**
CMPNN (Baseline)	0.958077 +/- 0.017582	0.915815 +/- 0.015748	0.640620 +/- 0.039283	0.851490 +/- 0.009426	***0.808721 +/- 0.020270***	0.582865 +/- 0.048626	0.969931 +/- 0.335350	0.583237 +/- 0.021075
GNNML3 (Spectral) (K=2)	0.761271 +/- 0.058912	0.431808 +/- 0.130315	0.558689 +/- 0.016746	0.742669 +/- 0.025536	0.707289 +/- 0.037594	1.364229 +/- 0.156225	3.520960 +/- 0.320290	1.128885 +/- 0.043017
CMPNN+ AttMaskedSM (v6)	**0.959694 +/- 0.014363**	0.901515 +/- 0.032833	0.638332 +/- 0.039486	0.849045 +/- 0.015064	0.787212 +/- 0.043480	**0.576424 +/- 0.060259**	**0.947268 +/- 0.313322**	0.625929 +/- 0.028989
CMPNN+ S2SPool	**0.960700 +/- 0.015843**	0.876158 +/- 0.071887	**0.650795 +/- 0.037669**	0.847987 +/- 0.011103	0.788879 +/- 0.035026	**0.579205 +/- 0.057359**	1.043572 +/- 0.409796	**0.578896 +/- 0.029028**
CMPNN+ GNNML3 (K=2)	0.957175 +/- 0.017800	0.913060 +/- 0.018448	***0.655724 +/- 0.039114***	0.846881 +/- 0.012181	0.784601 +/- 0.029780	**0.569737 +/- 0.072157**	***0.933951 +/- 0.250933***	**0.575096 +/- 0.016584**
CMPNN+ Bi-LSTM	**0.959280 +/- 0.022079**	**0.918462 +/- 0.018300**	0.640193 +/- 0.042409	***0.853141 +/- 0.010425***	0.779633 +/- 0.034216	**0.569297 +/- 0.042668**	0.989492 +/- 0.310800	**0.570735 +/- 0.030575**
CMPNN+Bi-LSTM+ EMB	***0.963301 +/- 0.015882***	***0.973677 +/- 0.008969***	**0.651875 +/- 0.040368**	0.850233 +/- 0.011002	0.797270 +/- 0.036407	***0.567632 +/- 0.062755***	**0.945355 +/- 0.279276**	***0.569006 +/- 0.026226***

Table 10: Number of winning tasks of CMPNN variants compared with the baseline (numbers within the parenthesis are the wining classification and regression tasks)

	CMPNN (Baseline)	GNNML3 (Spectral) (K=2)	CMPNN+ AttMaskedSM (v6)	CMPNN+ S2SPool	CMPNN+ GNNML3 (K=2)	CMPNN+ Bi-LSTM	CMPNN+ Bi-LSTM+EMB
Better	-	0	3(1,2)	4(2,2)	4(1,3)	5(3,2)	6(3,3)
Best	1	0	0	0	2(1,1)	1(1,0)	4(2,2)

In this thesis, we introduced and experimented with the variants of a graph neural network (GNN) with CMPNN as the core component for molecular property predictions. We introduced modern techniques such as edge attention and multimodal modules into or as a part of the model architecture.

From our experimental results, various techniques could help improve the overall performance of the baseline CMPNN for some datasets. For example, the introduced Bi-LSTM and GNNML3 beats all other models and got the best performance on 1 and 2, dataset(s), respectively, with an even better version of Bi-LSTM+EMB (embedding+Bi-LSTM) which got 4 best performance over the baseline CMPNN and all other variants. The use of edge attention also improved the performance of the baseline CMPNN but still did not win other techniques. Even though GNNML3 were made for graph isomorphism tasks, CMPNN+GNNML won 2 of the arguably difficult datasets which are SIDER (extreme class imbalances) and FreeSolv (super small dataset with small molecular graphs as stated in Table 8). Overall, our results demonstrate that the multimodal text modules could help improve the performance of the baseline CMPNN for molecular property prediction.



Chapter 5
Conclusion

This thesis focuses on improving the baseline CMPNN model for molecular properties prediction, specifically in the graph classification/regression task. The work begins with a comprehensive literature review covering various spatial GNN approaches such as DMPNN, CMPNN, GCN, and GAT. Additionally, the thesis revisits spectral GNNs and their procedures, highlighting the breakthrough GNNML3 model.

Building upon the existing literature, the thesis proposed two methods for enhancing the baseline CMPNN model. Firstly, we introduced a self-attention mechanism inspired by the Graph Attention Network (GAT) upon CMPNN. This self-attention allows the model to pay attention to the relevance of source node features to the target node features. Secondly, we developed a multimodal module to extract diverse information perspectives from the same dataset, which were then provided to the same MLP classifier/predictor. The proposed modules employed two different methods. The first method involved using a bidirectional LSTM and an embedding layer to embed SMILES representations, thereby capturing text-based features. The second method utilized GNNML3 for spectral feature extraction, enabling the extraction of graph frequencies, spectrums, and long-range dependencies.

The thesis evaluated the proposed methods through rigorous experimentation and compared their performance against other approaches. The combination of CMPNN, Bi-LSTM, and Embedding among the assessed models gave the best results.

This thesis contributes to the field of molecular properties prediction by improving the baseline CMPNN model and incorporating self-attention and multimodal techniques. The findings highlight the importance of extracting text features for more accurate predictions, and the proposed approach showcases its effectiveness through superior performance compared to alternative models.

REFERENCES

1. Zhenqin Wu, B.R., Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing and -.P.G. Stanford. *MoleculeNet Dataset Collection*. [cited 2022; MoleculeNet Datasets]. Available from: https://moleculenet.org/datasets-1.
2. Wieder, O., et al., *A compact review of molecular property prediction with graph neural networks.* Drug Discovery Today: Technologies, 2020. **37**: p. 1-12.
3. Jin, Z., et al., *GNNVis: A Visual Analytics Approach for Prediction Error Diagnosis of Graph Neural Networks*. 2020.
4. Gilmer, J., et al., *Neural Message Passing for Quantum Chemistry.* arXiv e-prints, 2017: p. arXiv:1704.01212.
5. Yang, K., et al., *Analyzing Learned Molecular Representations for Property Prediction.* Journal of Chemical Information and Modeling, 2019. **59**(8): p. 3370-3388.
6. Song, Y., et al. *Communicative Representation Learning on Attributed Molecular Graphs*. in *IJCAI*. 2020.
7. Rong, Y., et al., *Self-Supervised Graph Transformer on Large-Scale Molecular Data*. 2020: arXiv.
8. Vaswani, A., et al., *Attention Is All You Need*. arXiv.
9. Kipf, T.N. and M. Welling, *Semi-Supervised Classification with Graph Convolutional Networks.* arXiv e-prints, 2016: p. arXiv:1609.02907.
10. Veličković, P., et al., *Graph Attention Networks.* arXiv e-prints, 2017: p. arXiv:1710.10903.
11. Dagar, A. *Understanding Graph Attention Networks (GAT)*. Tuesday, Jan 21, 2020 20/11/2022]; Available from: https://dsgiitr.com/blogs/gat/.
12. Xu, K., et al., *How Powerful are Graph Neural Networks?* 2018: arXiv.
13. Wang, H., et al., *Equivariant and Stable Positional Encoding for More Powerful Graph Neural Networks.* arXiv e-prints, 2022: p. arXiv:2203.00199.
14. Muhammet, B., et al., *Breaking the Limits of Message Passing Graph Neural Networks*, in *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 2021.
15. Wu, Z., et al., *MoleculeNet: a benchmark for molecular machine learning.* Chemical Science, 2018. **9**(2): p. 513-530.
16. Weininger, D., *SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules.* Journal of Chemical Information and Computer Sciences, 1988. **28**(1): p. 31-36.
17. *RDKit: Open-Source Cheminformatics Software*. 21 Feb 2023]; Available from: https://www.rdkit.org.
18. Fey, M. and J.E. Lenssen, *Fast Graph Representation Learning with PyTorch Geometric.* arXiv e-prints, 2019: p. arXiv:1903.02428.

VITA

NAME	Kamol Punnachaiya
DATE OF BIRTH	13 April 1998
PLACE OF BIRTH	Bangkok
INSTITUTIONS ATTENDED	Chulalongkorn University
PUBLICATION	K. Punnachaiya, P. Vateekul and D. Wichadakul, "Multimodal Modules and Self-Attention for Graph Neural Network Molecular Properties Prediction Model" April 21st-23rd, 2023 11th International Conference on Bioinformatics and Computational Biology (ICBCB), Hangzhou, China, 2023.