8-8-2024

Hypergraph Learning: From Algorithms to Applications

Khaled Mohammed Saifuddin

Hypergraph Learning: From Algorithms to Applications

by

Khaled Mohammed Saifuddin

Under the Direction of Esra Akbas, Ph.D.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2024

ABSTRACT

Graphs are a general language for describing and modeling interconnected systems. To learn graph data, Graph Neural Networks (GNNs) have been introduced. However, traditional graph data structures often fall short of describing the higher-order complex relationships within these systems. Hypergraphs, with their natural ability to capture such higher-order relations, offer a promising alternative. Despite their potential, GNNs are inherently designed for simple graphs and do not extend naturally to hypergraphs, leaving a gap in effectively leveraging hypergraph structures.

To address this gap, Hypergraph Neural Networks (HyperGNNs) have been proposed. HyperGNNs offer enhanced capabilities to learn higher-order complex relationships beyond the scope of traditional GNNs. However, despite their potential, there remains a gap in effectively leveraging HyperGNNs for complex real-world problems due to limitations in current methodologies and applications. This dissertation aims to bridge this gap by developing and presenting new models for HyperGNN and examining their applications in real-world challenges. This work is built upon four pivotal studies, each emphasizing the development of novel HyperGNNs to tackle complex issues, especially in biomedical contexts, while also advancing methodologies to achieve superior outcomes.

The first study introduces a novel Hypergraph Neural Network model, `HyGNN` for drug-drug interaction prediction (DDI). By leveraging the SMILES strings of drugs, this approach first constructs a drug hypergraph presenting drugs as hyperedges and substructures of drugs as nodes. Then it utilizes a novel attention-based hyperedge encoder to generate drug representation for DDI prediction. **The second study** extends the application of HyperGNNs to sequence classification with the development of the Sequence Hypergraph Attention Network (`Seq-HyGAN`). This model captures complex structural similarities between sequences by defining higher-order relationships via common subsequences. By employing a novel hypergraph attention mechanism, `Seq-HyGAN` significantly improves the accuracy of sequence classification tasks, demonstrating the effectiveness of HyperGNNs in sequence data mining.

In the next 2 projects, we shift our focus to methodological advancements. Existing hypergraph transformers predominantly utilize semantic feature-based self-attention, resulting in the loss of the structural attributes of nodes and hyperedges. As a solution, **the third study** presents the Structure-aware Hypergraph Transformer (`SaHT`). `SaHT` model integrates both structural and spatial information into node representations, utilizing a structure-aware self-attention mechanism. By comprehensively considering both semantic and topological attributes, `SaHT` achieves notable improvements in node classification performance, thereby advancing the capabilities of hypergraph representation learning. Furthermore, traditional graph contrastive learning methods struggle to capture higher-order relationships inherent in complex data. **The fourth study** addresses this challenge by presenting `HyperGCL` that generates multiple hypergraph views from the input graph to preserve comprehensive higher-order structural and attribute information. By employing a learnable augmentation function, view-specific encoders, and a network-aware contrastive loss, this framework significantly enhances the performance of node classification tasks, establishing a new benchmark in contrastive learning for graph data. Collectively, these studies highlight the robustness and versatility of HyperGNNs in both practical applications and theoretical advancements.

INDEX WORDS: Hypergraph Neural Network, Hypergraph Transformer, Graph Contrastive Learning, DDI Prediction, Sequence Data Analysis

Hypergraph Learning: From Algorithms to Applications

by

Khaled Mohammed Saifuddin

DEDICATION

With love and immense appreciation, I dedicate this dissertation to my parents, Md Golam Hossain and Rokeya Begum, my siblings, and my wife. Your sacrifices, love, and constant encouragement have been my greatest source of inspiration and strength.

The path of doctoral research has shown me the wide and detailed world of computer science, revealing many important connections. This journey has taught me that true innovation comes not only from technical skills but also from strong determination, consistent hard work, and a constant desire to learn. Reflecting on this transformative experience, I realize that each data point tells a story, and every line of code contributes to a meaningful and lasting impact.

ACKNOWLEDGMENTS

My sincere thanks to Dr. Esra Akbas, my committee chair, for her mentorship, constructive feedback, and constant encouragement. Her wisdom has been a profound source of inspiration. I am deeply grateful for her patience and support throughout this journey, which has significantly shaped my academic growth. I am also indebted to Dr. Raj Sunderraman, Dr. Jonathan Shihao Ji, and Dr. Ugur Kursuncu for their willingness to serve on my committee and share their valuable knowledge.

I extend my gratitude to all members of DELab for fostering a supportive environment that has been crucial to my research progress. I am also grateful to the Department of Computer Science at Georgia State University and the National Science Foundation (NSF) for providing the essential resources and opportunities for this research.

I am deeply grateful to my Creator, Allah (SWT), for His blessings and guidance, and to the Prophet Muhammad (pbuh) for his teachings that illuminate the purpose of life. Finally, I thank my family and friends for their unwavering love, support, and encouragement. Your belief in me has been a constant source of strength and inspiration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background and Motivation

The field of machine learning has undergone significant transformations over the past few decades, with Graph Neural Networks (GNNs) emerging as a pivotal innovation. GNNs have demonstrated remarkable efficacy in modeling and learning from graph-structured data across various domains, including social networks [1], recommendation systems [2], and biological networks [3]. However, since traditional graphs are limited to capturing only dyadic relationships within a network, GNNs designed for these graphs fail to capture higher-order relationships—interactions involving more than two entities simultaneously. This limitation is significant in many real-world applications where complex relationships are the norm rather than the exception [4, 5].

Hypergraph Neural Networks (HyperGNNs) have been developed to address this limitation. Unlike traditional graphs, hypergraphs can model complex, multi-way relationships directly, offering a more expressive and comprehensive representation of data [6]. HyperGNNs extend the capabilities of GNNs by enabling the modeling of these higher-order interactions, thereby providing a more nuanced understanding of complex data structures [7, 8, 9, 10]. This advancement is particularly beneficial in domains such as biomedical research, where interactions between multiple entities (e.g., genes, proteins, drugs) are crucial for accurate modeling and analysis [11, 12].

This dissertation explores the development and application of HyperGNNs, presenting novel methodologies that enhance the expressivity and efficiency of these models. We investigate two key application areas of hypergraph learning: drug-drug interaction prediction and sequence classification. Additionally, we conduct two research studies focusing on improving the performance of HyperGNNs by designing a hypergraph transformer and graph contrastive learning from the hypergraph viewpoint. Each area addresses specific challenges and demonstrates the versatility and power of HyperGNNs in different contexts.

1.2 Overview of the Studies

1.2.1 Study 1: HyGNN for Drug-Drug Interaction Prediction

Drug-Drug Interactions (DDIs) pose significant risks in clinical treatments, often leading to Adverse Drug Reactions (ADRs) that can compromise patient safety [13, 14]. Traditional clinical trials are limited in scope and duration, often failing to identify all potential DDIs before drugs reach the market [15]. Computational models have thus become indispensable for early DDI detection. With the availability of public databases like DrugBank[1], STITCH[2], SIDER[3], PubChem[4], and KEGG[5], various computational models have been proposed to detect DDIs [16, 17].

These models often consider drug pairs' chemical structure SMILES similarity or binding properties [18]. SMILES, using ASCII characters, explicitly defines molecular structures. However, only a few substructures within a chemical structure are responsible for chemical

[1]https://go.drugbank.com/
[2]http://stitch.embl.de/
[3]http://sideeffects.embl.de/
[4]https://pubchem.ncbi.nlm.nih.gov/
[5]https://www.kegg.jp/

reactions between drugs, making the consideration of the entire structure potentially biased and undermining DDI prediction [19]. Moreover, to improve accuracy, current methods integrate multiple data sources to extract drug features such as side effects, target proteins, pathways, and indications [20, 15].

Network-based methods for DDI prediction construct drug networks based on known DDIs, typically considering dyadic relationships where each vertex represents a drug and each edge represents an interaction between drug pairs. Some methods also create heterogeneous graphs that include relationships between drugs and other biological entities to predict unknown drug interactions using various topological information. With advancements in GNNs, different models for DDI prediction have been proposed [17, 21], either by manually creating heterogeneous graphs from different resources or by building biomedical knowledge graphs from raw data sources like DrugBank [22, 23, 6]. These graphs represent entities such as drugs, proteins, and side effects as nodes, and their relationships as edges. While integrating multiple drug-centric information can enhance DDI prediction, it poses challenges due to the difficulty of data integration and interpretation, especially for new drugs in the early development stages, where information may be unavailable [24].

To overcome these limitations, we propose a hypergraph-based model that relies solely on the SMILES strings of drugs, which are universally available. Our method relies on the hypothesis that similar drugs behave similarly, are likely to interact with the same drugs, and two drugs are similar if they have similar substructures as functional groups in their SMILES strings [25, 26]. To depict the higher-order structural similarities between drugs, we construct a hypergraph where drugs are represented as hyperedges connecting multiple substructures as nodes derived from SMILES strings. This representation captures the intricate similarities between drugs more effectively than traditional graph-based models. Then we introduce a variant of Hypergraph Neural Network named `HyGNN`, which employs a novel hyperedge encoder to learn drug embeddings. Pairwise drug representations are then passed through decoder functions to predict a binary score indicating potential interactions. Our method demonstrates superior performance in predicting DDIs, especially for new drugs, without requiring extensive biomedical knowledge.

1.2.2 Study 2: Seq-HyGAN for Sequence Learning

Extracting meaningful features from sequences and devising effective similarity measures are crucial for sequence data mining tasks, particularly sequence classification. Neural networks (NN), especially recurrent neural networks (RNN) like LSTM and GRU, are commonly used to learn features capturing adjacent structural information [27, 28, 29]. However, these models may struggle to capture non-adjacent, high-order, and complex relationships present in sequence data. Recently, graphs have been explored for various sequence data classification tasks such as text classification [30], DNA-protein binding prediction [31], protein function prediction [32], and drug-drug interaction prediction [33]. Graphs, as sophisticated data structures, can effectively capture both local and global non-adjacent information within the data [34, 35]. State-of-the-art models for sequence-to-graph conversion fall into two main categories: order-based graphs and similarity-based graphs [36, 37, 38].

While existing graph models for sequence data have achieved great performance, they still face key challenges. Order-based models generate many large and sparse graphs, especially when dealing with long sequences or a large number of sequences, leading to increased computational and memory requirements. Similarly, similarity-based graphs encounter difficulties in calculating similarities between all pairs of sequences, especially for large datasets. Additionally, order-based graphs fail to capture relationships beyond intra-sequence connections, considering only dyadic relationships between nodes. Sequence data, however, may possess more complex relationships, such as triadic or tetradic relationships, which these models cannot capture. The selection of appropriate similarity measures is problematic, and using similarity values to define relationships between sequences can lead to information loss.

To address the challenges in sequence classification, we propose a novel Hypergraph Attention Network model, `Seq-HyGAN`. Our approach assumes that sequences sharing structural similarities tend to belong to the same classes and can be considered similar if they contain similar subsequences. To capture these structural similarities, we represent sequences in a hypergraph framework, where sequences are depicted as hyperedges connecting their respective subsequences as nodes. This allows us to create a single hypergraph encompassing all sequences in the dataset.

Unlike standard graphs where edges connect only two nodes, hyperedges in a hypergraph can connect an arbitrary number of nodes [6, 39, 40]. To enhance sequence representation and capture complex relationships, we introduce a three-level attention-based neural network architecture in `Seq-HyGAN`. Unlike regular neural networks (e.g., RNNs) that capture only local information, `Seq-HyGAN` captures both local (within sequences) and global (between sequences) information. Furthermore, while traditional GNNs are limited to message passing between two nodes, the hypergraph setting in `Seq-HyGAN` enables message passing between many nodes and between nodes and hyperedges, resulting in a more robust sequence representation.

1.2.3 Study 3: SaHT for Enhancing Node Representation

Inspired by the success of transformers in text [41], the transformer architecture has been extended to handle graph-structured data, resulting in graph transformers [42, 43, 44, 45, 46, 47]. These models aggregate weighted semantic-structural information from neighboring nodes, leveraging pairwise relational information to learn meaningful node and graph representations [48, 49]. However, real-world relationships often extend beyond pairwise interactions, exhibiting complex higher-order dynamics that standard graph transformers fail to capture.

Hypergraphs model higher-order interactions in complex systems by representing entities as nodes and their interactions as hyperedges. Despite their strength, there are few real-world hypergraph datasets, leading to the common practice of converting standard graphs into hypergraphs by aggregating nodes with similar semantics [7, 8, 9]. This method, however, can result in a loss of detailed structural information. GNNs have shown great performance in graph representation learning, but adapting them for hypergraphs remains challenging due to the complexity of hypergraph structures. Researchers have introduced hypergraph representation learning techniques, including hypergraph neural networks [7], hypergraph convolution networks [50], and hypergraph attention networks [9]. These methods aim to learn node and hyperedge embeddings by considering both topology and attributes, offering improved solutions for understanding complex relationships. Building on the success of transformers in graph data analysis, hypergraph transformer models have been developed [51, 52, 53, 54]. However, these models primarily focus on attribute-based semantic features, neglecting structural attributes.

To address these limitations, we propose a novel Structure-aware Hypergraph Transformer (`SaHT`) model. `SaHT` generates node representations through a new structure-aware self-attention mechanism, identifying the importance of nodes and hyperedges from both semantic and structural perspectives. First, we create hypergraphs from standard graphs, preserving structural information by considering nodes' higher-order relations. Our `SaHT` model operates on these hypergraphs. In the input layer, we introduce a learnable structure encoding scheme, which includes local structure encoding, centrality encoding, and uniqueness encoding to capture essential local and global structural information. Additionally, we use a learnable hypergraph Laplacian eigenvector as node positional information, enabling distance-aware spatial encoding within the hypergraph.

Given that hyperedges contain multiple nodes with varying degrees of structural and semantic importance, and nodes may belong to multiple hyperedges with differing levels of importance, we introduce a structure-aware self-attention mechanism comprising two layers. The Local Structure-Aware Node-to-Hyperedge Attention layer aggregates node representations into a hyperedge representation by emphasizing structurally and semantically significant nodes, with structural importance determined using the node-local clustering coefficient and node coreness. The Global Structure-Aware Hyperedge-to-Node Attention layer aggregates hyperedge representations into a node representation by highlighting structurally and semantically important hyperedges, with structural importance defined using the hyperedge density score and the hyperedge clustering coefficient. The learned node representations are then used for node classification.

1.2.4 Study 4: HyperGCL for Graph Contrastive Learning

Contrastive Learning (CL) has become a prominent paradigm in self-supervised learning, excelling across various domains like computer vision and NLP by bringing augmented views of the positive samples closer together in the representation space while pushing apart the embeddings of negative samples. This is typically achieved using a similarity metric and contrastive loss. Frameworks like SimCLR [55], MoCo-v2 [56], CLIP [57], and MetAug [58] have set new benchmarks in CL, often surpassing supervised methods [59, 60, 61, 62].

Inspired by this success, Graph Contrastive Learning (GCL) extends CL principles to graph-structured data, using GNNs to learn robust representations. GCL involves generating augmented views of the input graph and maximizing agreement between these views to enhance node and graph representations. Notable GCL frameworks include Deep Graph InfoMax (DGI) [63], Graphical Mutual Information (GMI) [64], MVGRL [65], and AD-GCL [66].

However, GCL models face significant challenges. They often focus on local structures to learn discriminative information, limiting their ability to capture higher-order global information [64, 67, 68]. Additionally, they rely on handcrafted graph-augmented views (e.g., node dropping, edge perturbation, attribute masking), which can result in the loss of critical structural and attribute information and make models inflexible and poorly adaptable to diverse data [69, 70, 71]. Moreover, GCL methods often apply contrastive losses designed for computer vision without addressing the differences between images and graphs [67, 69, 72]. This can overlook network structural information, contradict the homophily principle, and lead to high computational and memory costs due to the reliance on many negative samples.

To address these issues, we present `HyperGCL`, an Attribute-Structure aware Graph Contrastive Learning framework from a Hypergraph perspective. Hypergraphs naturally model complex systems and can capture hidden higher-order information from standard graphs. To extract different granularities of higher-order information, we design three hypergraphs from the input graph and its attributes. First, an attribute-driven hypergraph view groups semantically similar nodes into hyperedges, capturing semantic similarities but potentially losing detailed structural information. This is mitigated by creating two structure-infused hypergraph views: local structure-infused and global structure-infused, capturing different levels of structural information.

Instead of applying predefined augmentation, we employ an adaptive augmentation technique using a learnable Gumbel-Softmax function for each hypergraph view. This introduces controlled stochasticity, generating robust samples for contrastive learning and enhancing training diversity. Moreover, the learnable Gumbel-Softmax refines the constructed views by selectively highlighting important relationships within the hypergraph. Following this, we apply view-specific encoders to the augmented views. For attribute-driven hypergraph augmented views, we use the Hypergraph Attention Network (HyGAN) [10, 73] to learn node embeddings. HyGAN employs a two-layer attention network to identify semantically important nodes and hyperedges, generating the ultimate node embeddings. However, since HyGAN prioritizes semantic features, it may lose structural information when applied to structure-infused hypergraphs. To address this, we design Structure-aware HyGAN (SHyGAN), which also uses a two-layer attention network but incorporates node structure encodings and structural inductive biases to identify important nodes and hyperedges from both semantic and structural perspectives.

Instead of traditional contrastive losses like InfoNCE [74] or NT-Xent [55], we introduce a novel network-aware contrastive loss, NetCL. This loss extends NT-Xent by incorporating network topology as supervised signals to better define positive and negative samples in HyperGCL. Unlike NT-Xent, which forms only a single positive pair per anchor, NetCL supports multiple positive pairs for each anchor. These positives are drawn from the same node in different hypergraph views, the neighbors of the anchor within a hypergraph view or the input graph.

Nodes not fulfilling these conditions are considered negative instances, referred to as NegS. To address the computational expense of considering all negative instances, we propose two selective negative sampling strategies: distance-based and similarity-based. In distance-based negative sampling, we select the 'a' most distant negative samples from the anchor node in the graph view. In similarity-based negative sampling, we select the 'a' least similar negative samples from NegS.

Notation	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A hypergraph with the set of nodes $\mathcal{V}$ and the set of hyperedges $\mathcal{E}$
(m,n)	Numbers of elements in $\mathcal{V}$ and $\mathcal{E}$, respectively
A	Incidence matrix representing the hypergraph
hd	Hyperedge density
p_i^l	Representation of node v_i in layer l
q_j^l	Representation of hyperedge e_j in layer l
α, β	Nonlinear activation function
W	Trainable weight matrices
Δ	Attention map in the transformer network
$\theta, \psi, \zeta, \phi$	Learnable parameters
Ω	Node structural importance score
Υ	Hyperedge structural importance score
$\mathcal{H}^a$	Attribute-driven hypergraph view
$\mathcal{H}^l$	Local structure-infused hypergraph view
$\mathcal{H}^g$	Global structure-infused hypergraph view
$sim(), dis()$	Similarity and distance function, respectively

Table 1.1 Main symbols

1.3 Terminology

In this section, we give the necessary terminology and symbols to facilitate discussion in the rest of the dissertation. Table 1.1 lists these symbols.

First, we define a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \ldots, v_m\}$ is the set of nodes and $\mathcal{E} = \{e_1, \ldots, e_n\}$ is the set of hyperedges. Each hyperedge e_j is degree-free and consists of an arbitrary number of nodes. A hypergraph can be represented by an incidence matrix A where $A_{i,j} = 1$ if the node v_i is in the hyperedge e_j ($v_i \in e_j$) and $A_{i,j} = 0$ otherwise. The hyperedge density (hd) quantifies the proportion of nodes that a hyperedge consists of compared to the total number of nodes in the hypergraph. In HyperGNN, p_i^l denotes node v_i representation in layer l, and q_j^l is hyperedge e_j representation in layer l. The nonlinear activation function α and the LeakyReLU activation function β are used within the neural networks. The trainable weight matrices are denoted by W. The attention map in the transformer network is represented by Δ. Other learnable parameters $\theta, \psi, \zeta, \phi$

are used for various transformations and computations within the neural networks. Node structural importance score is denoted by Ω, and hyperedge structural importance score is denoted by Υ. Different views of the hypergraph are represented as $\mathcal{H}^a$ (attribute-driven hypergraph view), $\mathcal{H}^l$ (local structure-infused hypergraph view), and $\mathcal{H}^g$ (global structure-infused hypergraph view). The similarity function $sim()$ and distance function $dis()$ are used to measure similarities and distances between nodes, respectively.

CHAPTER 2
RELATED WORKS

2.1 Graph and Hypergraph Neural Networks

Graph Neural Networks (GNNs) are deep learning models designed to handle graph-structured data by aggregating information from a node's local neighborhood. Among the notable models is the Graph Convolutional Network (GCN), introduced by Kipf and Welling [75]. GCNs apply convolution operations using a spectral approach based on the graph Laplacian, enabling effective information propagation across nodes. In another approach, GraphSAGE, developed by Hamilton et al. [76], samples a fixed-size neighborhood for each node and learns an aggregation function to generate node embeddings. This method allows for generalization to unseen nodes and graphs, adding a layer of versatility to GNN applications. Graph Attention Networks (GATs), brought forward by Veličković et al. [77], use attention mechanisms to weigh the contributions of neighboring nodes. This approach enhances the model's ability to handle various graph types with more precision and adaptability. Meanwhile, Message Passing Neural Networks (MPNNs), conceptualized by Gilmer et al. [78], exchange messages between nodes and update their representations based on these interactions. This model provides a unifying framework for various GNN techniques, improving their capacity to capture complex relationships within graph data.

Hypergraph Neural Networks (HyperGNNs) extend GNNs to hypergraphs, capturing higher-order complex relations. Various models have been proposed in this area. HGNNs [7] and HyperGCN [79] were among the first to apply graph convolution to hypergraphs.

HGNNs handle complex data correlations using hyperedge spectral convolution and the clique expansion technique, while HyperGCN introduces a generalized hypergraph Laplacian to capture complex data relationships effectively. Another approach, MPNN-R [80], treats hyperedges as distinct nodes, linking a "hyperedge node" with all constituent nodes and transforming the hypergraph into a graph for direct use with GNNs. Motivated by the success of attention networks on graphs, HAN [9] and HyperGAT [10] leverage attention mechanisms for hypergraphs, enabling adaptive learning of node and hyperedge significance. Direct message passing on hypergraphs is employed by HyperSAGE [81] and UniGNN [82], which avoid information loss compared to HGNN. The AllSetTransformer framework [83] blends Deep Sets [84] and Set Transformers [85] with HyperGNNs to learn multiset functions, providing substantial modeling flexibility and enhancing performance in various tasks.

2.2 Drug-Drug Interaction Prediction

Many works have been proposed for the DDI prediction problem over the years. These can be categorized into similarity-based, classification-based, and network-based methods.

2.2.1 Similarity-based methods

Previous works assume that similar drugs have similar interaction profiles and define different similarities between drugs. Traditionally, pharmacological, topological, or semantic similarity based on statistical learning is utilized to predict DDIs. Vilar et al. [25] identify DDIs based on molecular similarities, representing each drug by a molecular fingerprint, a bit vector reflecting the presence/absence of a molecular feature. INDI, developed by [86], uses seven different drug-drug similarity measures learned from drug side effects, fingerprints, therapeutic effects, etc. Another vital research by [87] incorporates four different biological information (e.g., target, transporter, enzyme, and carrier) of drugs to measure the similarity of drug pairs.

2.2.2 Classification-based methods

Some models extract features of drugs from various biological entities and drug interaction information and apply different machine learning (ML) methods for DDI training [88, 19, 89, 90]. Davazdahemami and Delen [89] construct a graph containing both drug-protein and drug-side effect interactions and employ classification methods on the feature set, producing many similarities and centrality metrics. Luo et al. [90] propose a DDI prediction server that provides real-time DDI predictions based only on molecular structure, using a 611-dimensional docking vector for drug pair features. Ibrahim et al. [88] first extract different similarity features and employ logistic regression to pick the best feature; later, the best feature is used in six different ML classifiers to predict DDIs. Zheng et al. [19] propose DDI-PULearn to address the lack of negative samples, generating negative samples using one-class SVM and kNN before predicting DDIs.

2.2.3 Network-based methods

Last decade, network-based models got great attention for drug-related problems. Some researchers construct a drug network using known DDI where drugs are nodes and interacting drugs are connected by a link [91]. Moreover, heterogeneous information networks leveraging different biomedical entities, such as proteins, side effects, pathways, etc., are also used to address similar problems [92]. As a different model, [93] constructs a molecular graph for each drug from its SMILES representation. Moreover, existing network-based models often extract drug embedding and directly learn latent node embedding using various embedding methodologies. As a result, their capacity to obtain specific neighborhood information on any organization in KG is restricted.

2.2.4 GNN-based methods

Recently, GNNs have shown promising performance in different fields, including drug discovery [94], drug abuse detection [30], and drug-drug interaction [95, 33, 93]. Decagon [17] creates a knowledge graph based on protein-protein, drug-drug, and drug-protein interactions and uses a relational graph convolutional neural network for predicting multi-relational links in multimodal networks. CASTER [96] develops a dictionary learning framework for predicting DDIs based on drug chemical structures, outperforming numerous deep learning approaches such as DeepDDI [97] and molVAE [98]. Bumgardner et al. [33] construct a drug network where two drugs are connected if they share common chemical substructures, applying different GNN models to predict interactions.

2.3 Sequence Classification

Various studies have been carried out on the problem of sequence classification, broadly categorized into three types of methods: ML-based, Deep Learning (DL)-based, and GNN-based.

2.3.1 Machine learning and deep learning-based methods

ML-based methods first generate a feature vector using kernel functions such as the k-spectrum kernel [99] and local alignment kernel [100]. These feature vectors are then used with ML classifiers for sequence classification tasks. Applications of DL-based methods, especially recurrent neural networks (RNN) such as LSTM and GRU, are commonly used to learn features capturing adjacent structural information for sequence data [27, 28, 101, 29]. Some studies use a single DL method, while others create hybrid models by combining different DL techniques.

2.3.2 Network-based methods

Network-based models have also been explored to analyze sequence data [102, 103]. A common approach for representing genome sequences in a network is the De-Bruijn graph [104]. To construct a De-Bruijn graph, the k-mer method is applied to sequence input, generating k-mer tokens as nodes. Subsequent k-mers with overlapping $k-1$ positions are connected with edges to construct the graph.

2.3.3 GNN-based methods

GNNs have exhibited great performance in different research areas, such as DDI prediction [92] and image classification [105]. GCN, a popular GNN variant, has also been applied for sequence data analysis [106, 107]. In [108], authors apply GCN for text classification by creating a heterogeneous text graph including document and word nodes from the whole corpus. The same architecture is applied for DNA-protein binding prediction from sequential data [31], where networks include sequence and token nodes extracted using the k-mer method.

2.4 Graph and Hypergraph Transformer

2.4.1 State-of-the art graph transformer model

Vaswani et al. introduce the transformer model in [41], highlighting the self-attention mechanism's capability to capture long-range dependencies in sequential data. This innovation inspired the development of graph transformers, which extend self-attention to handle graph-structured data. Dwivedi et al. advance this field with a graph transformer [109] that incorporates graph connectivity as an inductive bias, ensuring nodes attend only to their neighbors, utilizing graph Laplacian eigenvectors for node positional encoding. Gophormer [49] adopts a different approach by extracting ego-graphs as transformer inputs rather than processing the entire graph. It addresses the challenges of incorporating structural data into the transformer model through a proximity-enhanced attention mechanism [49] and by integrating centrality, spatial, and edge encoding techniques [42].

2.4.2 State-of-the art hypergraph transformers model

While transformers for standard graphs have seen considerable progress, a gap remains in developing transformers specifically for hypergraphs. Recent research in hypergraph transformers [51, 110, 52, 111, 54] shows promise but tends to focus predominantly on the attribute-based semantic features of nodes and hyperedges within the self-attention module, while often overlooking crucial structural information. Notably, models in [51, 110] are limited to meta-path-guided heterogeneous hypergraphs, further narrowing their scope.

2.5 Graph Contrastive Learning

2.5.1 Contrastive learning

Contrastive Learning (CL) is a widely used self-supervised learning technique, especially popular in computer vision and NLP. By optimizing a contrastive loss function, CL aims to learn an embedding space where samples from the same instance are positioned closer together, while samples from different instances are pushed apart. Notable CL methods in computer vision include SimCLR [55], MoCo [56], and BYOL [112]. In NLP, CL models such as SimCSE [61] and DeCLUTR [113] have been effectively utilized.

2.5.2 State-of-the art graph contrastive learning models

Graph Contrastive Learning (GCL) extends the principles of CL to GNNs to effectively capture both structural and attribute information in graph-structured data. DGI [114], inspired by Deep InfoMax [115], learns node representations by maximizing the mutual information between local graph patches and the global graph summary, thereby capturing global information often overlooked by traditional GCNs. GRACE [116], inspired by SimCLR [55], creates two augmented graph views by uniformly perturbing nodes, edges, and features, and learns node representations by contrasting the same node across these views. Thakoor et al. [70] generate two augmented views by masking node features and edges using different functions. GraphCL [69] introduces four types of augmentation techniques: node dropping, edge perturbation, attribute masking, and subgraph extraction. Similarly, CSSL [71] presents four graph alteration techniques, including node insertion/deletion and edge insertion/deletion, to generate augmented views.

Rather than randomly perturbing graph structures and features, MVGRL [65] uses a graph diffusion technique to generate an augmented view and contrasts it with the input graph. SCGDN [117] creates a k-NN graph as an augmented view using node attributes and contrasts it with the input graph. Existing GCL models often use contrastive loss functions from computer vision, treating the same node across different views as a positive sample and all other nodes, including neighbors, as negative samples [67, 69, 72]. This approach contradicts the homophily assumption, where a node and its neighbors often share the same label, and creates a large pool of negative samples, making the learning process computationally expensive.

CHAPTER 3

HYGNN: DRUG-DRUG INTERACTION PREDICTION VIA HYPERGRAPH NEURAL NETWORK

3.1 Introduction

In this paper, we present a novel GNN-based approach for DDI prediction that relies solely on the SMILES string of drugs, which is available for all drugs. Our method is based on the hypothesis that similar drugs behave similarly and are likely to interact with the same drugs, with similarity determined by common substructures in their SMILES strings [25, 26]. Identifying these similarities is challenging, so we represent drugs in a hypergraph framework, where drugs are hyperedges connecting multiple substructures as nodes. A hypergraph differs from a regular graph as hyperedges can connect an arbitrary number of nodes [6, 7, 40, 39].

After constructing the hypergraph, we develop a Hypergraph Neural Network, `HyGNN` to learn DDIs by generating representations of drugs as hyperedges. `HyGNN` follows an encoder-decoder architecture: the hyperedge encoder generates drug embeddings, and these pairwise drug representations are processed through decoder functions to predict a binary interaction score for each drug pair.

The main contributions of this paper are summarized as follows:

- **Hypergraph Construction from SMILES Strings:** We construct a hypergraph to depict drug similarities. Substructures extracted from SMILES strings are represented as nodes, while drugs, consisting of unique substructures, are represented as hyperedges. This hypergraph captures higher-order connections between substructures and drugs, aiding in defining complex similarities and enhancing GNN models' ability to learn robust drug representations.

- **Hypergraph GNN:** To learn and predict DDIs, we propose `HyGNN`, which includes a novel hyperedge encoder and a decoder. The encoder has two layers: the first generates node embeddings by aggregating hyperedge embeddings, and the second generates hyperedge (drug) embeddings by aggregating node embeddings. An attention mechanism is used to identify significant substructures for drugs and chemical reactions. The decoder predicts DDIs by taking pairwise drug representations as input. Our method exclusively uses chemical structure data from SMILES strings, making it applicable to any drug, including new ones, without needing additional information like side effects or existing DDIs.

- **Extensive Experiments:** We conduct extensive experiments comparing our model with state-of-the-art models. The results show that our method significantly outperforms all baselines across various accuracy measures. Additionally, case studies demonstrate that our model can discover new DDIs for existing and new drugs.

3.2 `HyGNN` model for DDI

In this section, we first define our DDI prediction problem and then summarize the preliminaries, model, and settings (Section 3.2.1). Then we explain our hypergraph construction step with substructures extraction from Drugs (Section 3.2.2). After that, we introduce our proposed `HyGNN` model with attention-based encoder and decoder layers (Section 3.2.3).

3.2.1 Problem Formulation

Our goal is to develop a computational model that takes a drug pair (D_x, D_y) as input and predicts whether there exists an interaction between this drug pair. Each drug is represented by the SMILES string. SMILES is a unique chemical representation of a drug that consists of a sequence of symbols of molecules and the bonds between them.

Most of the graph-based existing DDI methods consider a dyadic relationship between drugs. This simple graph type considers an edge that can connect a maximum of two objects. However, there could be a more complex network in real life where an arbitrary number of nodes may interact as a group, so they could be connected through a hyperedge (i.e., triadic, tetradic, etc.). A hypergraph can be used to formulate such a complex network. A formal definition of the hypergraph is given below.

Hypergraph: *A hypergraph is a special kind of graph defined as* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *where* $\mathcal{V} = \{v_1, ..., v_m\}$ *is the set of nodes and* $\mathcal{E} = \{e_1, ..., e_n\}$ *is the set of hyperedges. Each hyperedge* e_j *is degree-free and consists of an arbitrary number of nodes. Like the adjacency matrix of a regular graph, a hypergraph can be denoted by an incidence matrix* A *with* $A_{i,j} = 1$ *if the node* i *is in the hyperedge* j *as* $v_i \in e_j$ *and* $A_{i,j} = 0$ *otherwise.*

In this paper, we construct a hypergraph network of drugs where each drug is a hyperedge, and the chemical substructures of drugs are the nodes. The chemical structures of a drug can be obtained from the SMILES, a unique chemical representation of a drug. We design a novel hypergraph neural networks model as an encoder-decoder architecture to accomplish the DDI prediction task. The encoder part exploits an attention mechanism to learn the

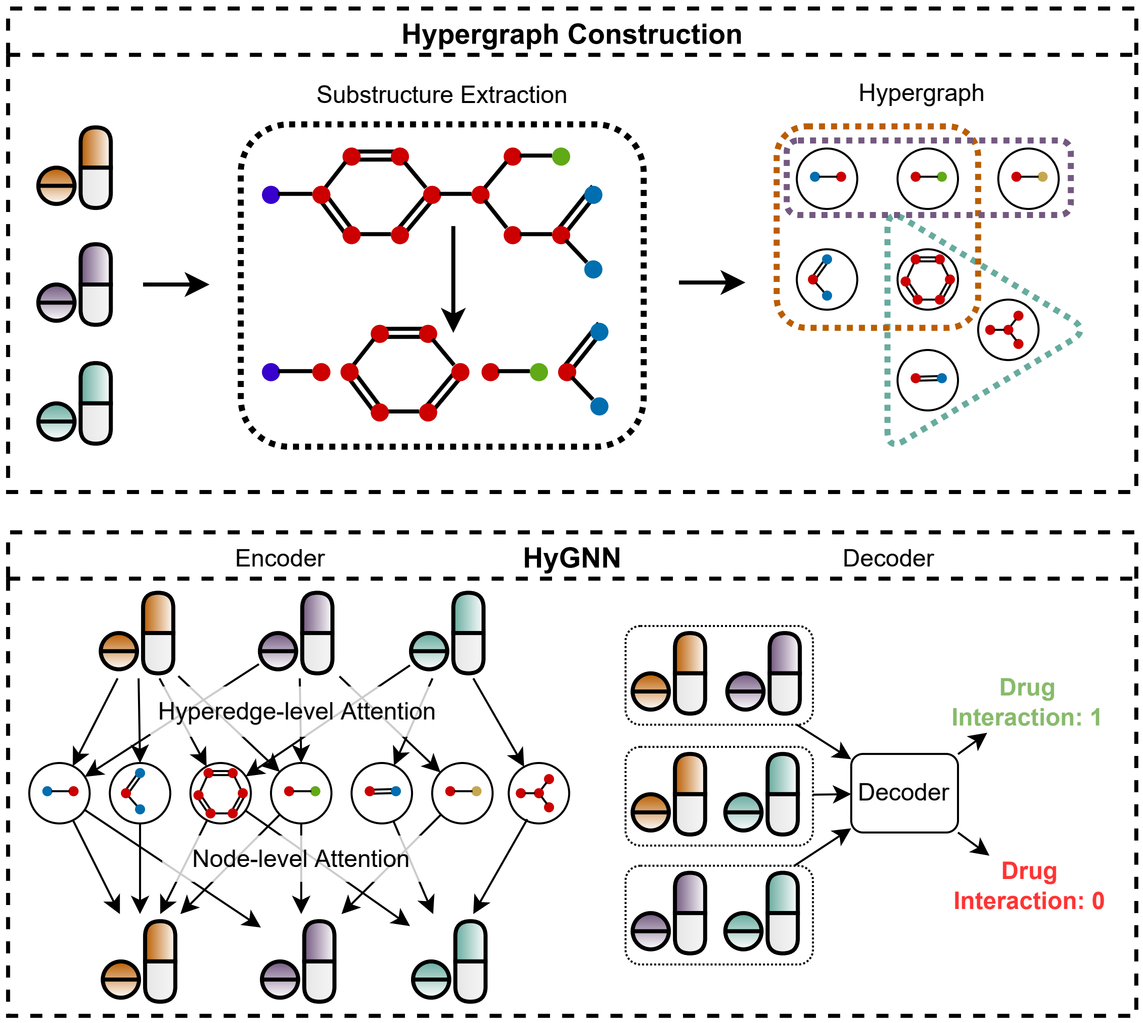


Figure 3.1 System architecture of the proposed method. The First step is to construct a hypergraph network of drugs where each drug is a hyperedge, and frequent chemical substructures of drugs are the nodes. The second step is to design a hypergraph neural network (`HyGNN`) model with an attention-based encoder for hyperedge (drug) representation learning and decoder for DDI learning.

representations of hyperedges (drugs) by giving attention to edges and nodes (substructures). The decoder predicts the interaction between drug pairs using latent learned drug features. The whole system is trained in a semi-supervised fashion. The functional architecture of this paper is shown in Figure 3.1. Our proposed model consists of two steps:

1. Hypergraph construction from SMILES,
2. DDI prediction with hypergraph neural networks
   (a) Encoder: Drug (Hyperedge) representation learning
   (b) Decoder: DDI prediction

3.2.2 Drug Hypergraph Construction

We construct a hypergraph to depict the structural similarities among drugs. At first, we decompose all the drugs' SMILES into a set of substructures. In our drug hypergraph, these substructures are used as nodes. Moreover, each drug with a certain number of substructures is represented by a hyperedge. Drugs as hyperedges may connect with other drugs employing shared substructures as nodes. This hypergraph represents the higher-level connections of substructures and drugs, which may help to define complex similarities between chemical structures and drugs. Also, this helps us to learn better representation for drugs with GNN models with the passing message not only between 2 nodes but between many nodes and also between nodes and edges.

Substructures can be generated by utilizing different algorithms such as ESPF [118], k-mer[119], strobemers [120], etc. In this project, we use ESPF and k-mer to see the effect of substructures on the results. While k-mer use all extracted substructures, ESPF selects the most frequent ones. Algorithm 1 briefly shows the hypergraph construction steps.

ESPF: Like the concept of sub-word units in the natural language processing (NLP) domain, ESPF is a powerful tool that decomposes sequential structures into interpretable

Algorithm 1: Drug Hypergraph Construction

```
Input: $SMILES\_strings$
Output: Hypergraph incident matrix: $A$
Call $Substructure\_Decom(SMILES\_strings)$;   /* Substructure_Decom() could
 be ESPF or $k$-mer that decomposes SMILES into substructures. It
 returns a list of unique substructures and drug dictionary that will
 be used in the following for loop */
for each Substructure in Substructure_list do
    if Substructure is in Drug_dict[SMILES] then
        $A[i, j] = 1$ ;            /* i,j is the id of substructure and drug,
         respectively. */
    end
end
Output: $A$, Hypergraph incident matrix
```

Algorithm 2: Explainable Substructure Partition Fingerprint (ESPF)

```
Input: Set of initial SMILES tokens $S$ as atoms and bonds, set of tokenized
 SMILES strings $TS$, frequency threshold $\alpha$, and size threshold $L$ for $S$.
for $t = 1 \ldots, L$ do
    $(S1, S2)$, $f \leftarrow$ scan $TS$ ;      /* $(S1, S2)$ is the frequentest consecutive
     tokens. */
    if $f < \alpha$ then
        break ;              /* $(S1, S2)$'s frequency lower than threshold */
    end
    $TS \leftarrow$ find $(S1, S2) \in TS$, replace with $(S1S2)$ ;     /* update $TS$ with the
     combined token $(S1S2)$ */
    $S \leftarrow S \cup (S1S2)$ ;      /* add $(S1S2)$ to the token vocabulary set $S$ */
end
Output: $TS$, the updated tokenized drugs; $S$, the updated token vocabulary set.
```

functional groups. They consider that a few substructures are mainly responsible for drug chemical reactions, so they extract frequent substructures as influential ones. The ESPF algorithm is shown in Algorithm 2. Given a set of drug SMILES strings S, ESPF finds the frequent repetitive moderate-sized substructures from the set and replaces the original sequence with the substructures. If the frequency of each substructure in S is above a predefined threshold, it is added to a substructure list as a vocabulary. Substructures appear in this list in the most to least frequent order. We use this vocabulary list as our nodes and decompose any drug into a sequence of frequent substructures concerning those. For any given drug, we partition its SMILES in order of frequency, starting from the highest frequency. An example of partitioning a SMILES of a drug, DB00226, is as follows.

$$\texttt{NC(N)=NCC1COC2(CCCCC2)O1}$$

$$\Downarrow$$

$$\underline{\texttt{N}}\ \underline{\texttt{C(N)}}\ \underline{\texttt{=N}}\ \underline{\texttt{CC1}}\ \underline{\texttt{CO}}\ \underline{\texttt{C2}}\ \underline{\texttt{(CCC}}\ \underline{\texttt{CC2)}}\ \underline{\texttt{O1}}.$$

k-mer: *k*-mer is a tool to decompose sequential structures into subsequences of length *k*. It is widely used in biological sequence analysis and computational genomics. Similar to n-gram in NLP, a *k*-mer is a sequence of *k* characters in a string (or nucleotides in a DNA sequence). To get all *k*-mers from a sequence, we need to get the first k characters, then move just a single character to start the next *k*-mer, and so on. Effectively, this will create sequences that overlap in k-1 positions. Pseudocode for *k*-mer is shown in Algorithm 3.

For a sequence of length l, there are $l - k + 1$ numbers of k-mers and n^k total possible number of k-mers, where n is the number of monomers. k-mers are like words of a sentence. k-mers help to bring out semantic features from a sequence. For example, for a sequence `NCCO`, monomers: {N, C, and O}, 2-mers: {NC, CC, CO} , 3-mers: {NCC, CCO}.

Algorithm 3: k-mer

```
Input: $SMILES\_strings$, size threshold $k$
Substructure_list: []
Drug_dict:{}
for each $SMILES$ in $SMILES\_strings$ do
    Lst=[]
    for $x$ in range (l-k+1) do
        /* $l$ is the length of $SMILES$ */
        $C = SMILES[x : x + k]$
        $Lst.append(C)$
        $Substructure\_list.append(C)$
    end
    $Drug\_dict[SMILES] = Lst$
end
Output: Drug_dict, Substructure_list
```

3.2.3 Hypergraph Neural Network (HyGNN) for DDI Prediction

We design a Hypergraph Neural Network named (HyGNN) for DDI prediction. HyGNN includes an encoder, which generates the embedding of drugs, and a decoder that uses the embedding of drugs from the encoder to predict whether a drug pair interacts or not.

3.2.3.1 Drug Representation learning via HyGNN - Encoder

To detect interacting pairs of drugs, we need features of drug pairs and, thus, features of drugs that encode their structure information. To generate features of drugs, we propose a novel *hyperedge encoder*. It creates d' dimensional embedding vectors for hyperedges (drugs) instead of nodes as in regular GNN models. Given the edge feature matrix, $F \in R^{|\mathcal{E}|*d}$ and incidence matrix $A \in R^{|\mathcal{V}|*|\mathcal{E}|}$, the encoder of the HyGNN generates a feature vector of d' dimension through learning a function Z. Any layer (e.g., $(l+1)^{th}$ layer) of HyGNN can be expressed as

$$F^{l+1} = Z(F^l, A^T). \tag{3.1}$$

We consider the *hyperedge encoder* with a memory-efficient self-attention mechanism. It consists of two different levels of attention: (1) hyperedge-level attention, (2) node-level attention.

While hyperedge-level attention aggregates the hyperedge information to get the representation of nodes, node-level attention layer aggregates the connected vertex information to get the representation of hyperedges. In general, we define the `HyGNN` attention layers as

$$p_i^l = \mathbf{A_E}^l(p_i^{l-1}, q_j^{l-1} | \forall e_j \in \mathcal{E}_i), \tag{3.2}$$

$$q_j^l = \mathbf{A_V}^l(q_j^{l-1}, p_i^l | \forall v_i \in e_j) \tag{3.3}$$

where $\mathbf{A_E}$ is an edge aggregator that aggregates features of hyperedges to get the representation p_i^l of node v_i in layer-l and $\mathcal{E}_i$ is the set of hyperedges that are connected to node v_i. Similarly, $\mathbf{A_V}$ is a node aggregator that aggregates features of nodes to get the representation q_j^l of hyperedge e_j in layer-l and v_i is the node that connects to hyperedge e_j.

Hyperedge-level attention: In a hypergraph, each node may belong to multiple numbers of edges. However, the contribution of hyperedges to a node may not be equal. That is why we design an attention mechanism to highlight the crucial hyperedges and aggregate their features to compute the node feature p_i^l of node v_i. With the attention mechanism, p_i^l is defined as

$$p_i^l = \alpha\left(\sum_{e_j \in E_i} Y_{ij} W_1 q_j^{l-1}\right) \tag{3.4}$$

where α is a nonlinear activation function, W_1 is a trainable weight matrix that linearly transforms the input hyperedge feature into a high-level, $\mathcal{E}_i$ is the set of hyperedges connected to node v_i, and Y_{ij} is the attention coefficient of hyperedge e_j on node v_i. The attention coefficient is defined as

$$Y_{ij} = \frac{\exp(\mathbf{e_j})}{\sum_{e_k \in E_i} \exp(\mathbf{e_k})} \tag{3.5}$$

$$\mathbf{e_j} = \beta(W_2 q_j^{l-1} * W_3 p_i^{l-1}) \tag{3.6}$$

where β is a LeakyReLU activation function, W_2, W_3 are the trainable weight matrices, and $*$ is the element-wise multiplication.

Node-level attention: Each hyperedge in a hypergraph consists of an arbitrary number of nodes. However, the importance of nodes in a hyperedge construction may not be the same. We design a node-level attention mechanism to highlight a hyperedge's important nodes and aggregate their features accordingly to compute the hyperedge feature q_j^l of hyperedge e_j. With the attention mechanism, q_j^l is defined as

$$q_j^l = \alpha\left(\sum_{v_i \in e_j} X_{ji} W_4 p_i^l\right) \tag{3.7}$$

where W_4 is a trainable weight matrix, and X_{ji} is the attention coefficient of node v_i in the hyperedge e_j. The attention coefficient is defined as

$$X_{ji} = \frac{\exp(\mathbf{v_i})}{\sum_{v_k \in e_j} \exp(\mathbf{v_k})} \tag{3.8}$$

$$\mathbf{v_i} = \beta(W_5 p_i^l * W_6 q_j^{l-1}) \tag{3.9}$$

where v_k is the node that belongs to hyperedge e_j, W_5, W_6 are the trainable weight matrices.

Our hyperedge encoder model works based on these two attention layers that can capture high-order relations among data. Given the input hyperedge features, we first gather them to get the representation of nodes with hyperedge-level attention, then we gather the obtained node features to get the representation of hyperedges with node-level attention.

3.2.3.2 ***DDI prediction - Decoder***

After getting the representation of drugs from the encoder layer, our target is to predict whether a given drug pair interacts or not. To accomplish this target, we design a decoder.

Given the vector representations (q_x, q_y) of drug pairs (D_x, D_y) as input, the decoder assigns a score, $p_{x,y}$ to each pair through a decoder function defined as

$$p_{x,y} = \gamma(q_x, q_y) \tag{3.10}$$

We use two different types of decoder functions:

MLP: After concatenating the features of drug pairs, we pass it through a multi-layer perceptron (MLP), which returns a scalar score for each pair

$$\gamma(q_x, q_y) = f_2(f_1(q_x \parallel q_y)) \tag{3.11}$$

where f_1 and f_2 are two different layers of MLP, and $||$ is the concatenation operation.

Dot product: We compute a scalar score for each edge by performing element-wise dot product between features of drug pairs using

$$\gamma(q_x, q_y) = q_x \cdot q_y \tag{3.12}$$

Afterward, we pass the decoder output through a sigmoid function $\sigma(\gamma(q_x, q_y))$ that generates predicted labels, Y', within the range 0 to 1. Any output value closer to 1 implies a high chance of interaction between two drugs.

3.2.3.3 ***Training the whole model***

We consider the DDI prediction problem as a binary classification problem predicting whether there is an interaction between drug pairs or not. As a binary classification problem, we train our entire encoder-decoder architecture using a binary cross-entropy loss function defined as

$$loss = -\sum_{i=1}^{N} \Big(Y_i \log Y_i^{'} + (1 - Y_i) \log(1 - Y_i^{'}) \Big) \tag{3.13}$$

where N is the total number of samples, Y_i is the actual label, and $Y_i^{'}$ is the predicted label.

3.2.4 Complexity

Our method is highly efficient with paralyzing across the edges and nodes [121]. The hyperedge encoder generates $d^{'}$ dimensional embedding vector for each hyperedge with a given initial feature as d dimensional vector using two-level attention. Hence, the time complexity in the encoder part is the cumulative complexity of attention layers. According to equation 3.4, the complexity in the hyperedge-level attention can be expressed as $O(|\mathcal{E}|dd^{'} + |\mathcal{V}|Dd^{'})$, where D is the average degree of nodes. Similarly, the complexity in the node-level attention can be expressed as: $O(|\mathcal{V}|dd^{'} + |\mathcal{E}|Bd^{'})$, where B is the average degree of hyperedges.

3.3 Experiment

In this section, we evaluate our proposed `HyGNN` model for DDI prediction with extensive experiments on two different datasets. We use F1, ROC-AUC, and PR-AUC accuracy metrics to compare our model's performances with the state-of-art baseline models. Making DDI predictions for new drugs could be more challenging than existing drugs. Therefore, we assess our model's performance for both new and existing drugs as well. First, we describe our datasets, TWOSIDES and DrugBank, then we explain our experiments, and present and analyze our results.

3.3.1 Dataset

We evaluate the proposed model using two different sizes of datasets. One is a small dataset, and another one is a large dataset. (1) **TWOSIDES** is our small dataset. **TWOSIDES** was created using data from adverse event reporting systems. Common adverse effects,

Dataset	# of Drug	# of DDI
TWOSIDES	645	63473
DrugBank	1706	191402

Table 3.1 Statistics of Dataset

such as hypotension and nausea, occur in more than a third of medication combinations, but others, such as amnesia and muscular spasms, occur in only a few. We extract 645 approved drugs' information from **TWOSIDES**. Each drug is linked to its chemical structure (SMILES). There are 63,473 DDI positive labels for the selected drugs. (2) **DrugBank** is our large dataset and it is the largest dataset for drugs that is publicly available. It is a drug knowledge database that includes clinical information about drugs, such as side effects and (DDIs). **DrugBank** also includes molecular data, such as the drug's chemical structure, target protein, and so on. From **DrugBank**, we retrieve information on 1706 approved drugs along with their SMILES strings and 191,402 DDI information. Both datasets are publicly available on Therapeutics Data Commons (TDC) [1]. The first unified platform, TDC, was launched to comprehensively access and assess machine learning across the entire therapeutic spectrum.

All known DDIs in both datasets are our positive samples. However, to train our model, we need negative samples as well. Therefore, we randomly sample a drug pair from the complement set of positive samples for each positive sample. Thus, we ensure a balanced dataset of equally positive and negative samples for an individual dataset.

We apply the ESPF algorithm and k-mer separately to extract the substructures from the SMILES string of drugs. For ESPF, we notice that when a lower frequency threshold is set,

[1]https://tdcommons.ai/

ESPF	$\|N\|$	k**-mer**	$\|N\|$
5	555	3	822
10	324	6	7025
15	249	9	14002
20	208	12	17351
25	187	15	18155

Table 3.2 Number of Nodes (N) in the Hypergraph based on parameters of the methods, ESPF, and k-mer, for TWOSIDES Dataset

ESPF	$\|N\|$	k**-mer**	$\|N\|$
5	1266	3	1296
10	729	6	11849
15	550	9	29443
20	462	12	43634
25	400	15	51315

Table 3.3 Number of Nodes (N) in the Hypergraph based on parameters of the methods, ESPF, and k-mer, for DrugBank Dataset

it generates many substructures, some of which may be unimportant. However, when a more significant threshold value is set, it generates fewer substructures and may lose some critical substructures. These substructures are used as nodes in the hypergraph. To examine the impact of the frequency threshold and thus the number of nodes in the hypergraph learning, we choose five different threshold values from 5 to 25. For k-mer, we notice that typically with the increment of k, the number of substructures (i.e., nodes) also increases. Similarly, to examine the impact of the k and thus the number of nodes in the hypergraph learning, we choose five different values of k from 3 to 15. A statistic of both datasets is shown in Table 3.1. The number of nodes for different threshold values of ESPF and k-mer is given in Table 3.2 and Table 3.3 for each dataset.

Parameter	Values
Learning rate	1e-2, 5e-2, 1e-3, 5e-3
Hidden units	32, 64, 128
Dropout	0.1, 0.5
Weight decay	1e-2, 1e-3

Table 3.4 Hyper-parameter Settings

3.3.2 Parameter Settings

Each dataset is randomly split into three parts: train (80%), validation (10%), and test (10%). We repeat this five times and report the average performances in terms of F1-score, ROC-AUC, and PR-AUC. The optimal hyper-parameters are obtained by grid search based on the validation set. The ranges of grid search are shown in Table 3.4.

We employ a single-layer `HyGNN` having two levels of attention. We use a LeakyReLU activation function in the encoder side and a ReLU activation function in the MLP predictor of the decoder side. During training, we simultaneously optimize the encoder and decoder using adam optimizer. Each model is trained for 2000 epochs with an early stop if there is no change in validation loss for 200 consecutive epochs.

For the baselines in subsection: 3.3.3, each GNN model is used as a two-layer architecture. All other parameters of each GNN are set by following their sources. For DeepWalk and node2vec, the walk length, number of walks, and window size are set to 100, 10, and 5, respectively. We use Logistic Regression as a simple ML classifier.

3.3.3 Baselines

We compare our model performance with different types of state-of-the-art models. We categorize the baseline models into five groups based on the data representation and methodology.

1. Random walk-based embedding (RWE) on DDI graph We construct a regular graph based on the drug interaction information called a DDI graph. Drugs are represented as nodes, and two drugs share an edge if they interact. After constructing the graph, we apply the random walk-based graph embedding methods to get the representations of drugs. DeepWalk [122] and Node2vec [123] are two well-known graph embedding methods. They are both based on a similar mechanism of 'walk' on the graph traversing from one node to another. We apply DeepWalk and Node2Vec on the DDI graph and generate the embedding of nodes. Afterward, we concatenate drug embeddings to get the drug pair features and feed that into a machine learning classifier for binary classification.

2. GNN on DDI graph: After constructing the DDI graph as explained above, we apply three different GNN models with unsupervised settings; graph convolution network (GCN) [75], graph attention network (GAT) [121], and GraphSAGE [76] to get the representations of drugs. These GNN models are obtained from DGL[2]. After getting the representations of drugs, we concatenate them pair-wise and use them as the features of drug pairs in the ML classifier for binary classification.

3. GNN on substructure similarity graph (SSG) We follow [33] to create the substructure similarity graph (SSG). We construct an edge between two drugs if they have at least a predefined number of common substructures. We apply the ESPF algorithm to the SMILES strings of drugs to get the frequent substructures. Afterward, we apply three different GNN

[2]https://docs.dgl.ai/

models, GCN, GAT, and GraphSAGE, to the constructed graph to get the representations of drugs. The drug representations are then concatenated pair-wise and fed into a classifier to predict the DDI.

4. CASTER We apply the Caster algorithm [96] for DDI prediction. It takes SMILES strings as input and employs frequent sequential pattern mining to discover the recurring substructures. They use the ESPF algorithm to extract frequent substructures. Then, they generate a functional representation for each drug using those frequent substructures. Further, the functional representation of drug pairs is used to predict DDIs. We reproduce CASTER results for our datasets.

5. Decagon Decagon [17] uses a multi-modal graph consisting of protein-protein interactions and drug-protein target interactions for DDI prediction. It has an encoder-decoder architecture. The encoder exploits GCN to generate the representation of drugs by embedding all drug interactions with other entities in it. Then, the decoder takes drug pair representations as input and predicts DDIs with an exact side effect. The same TWOSIDES drug-drug interactions network is used in Decagon. That is why we directly compare our model performances with their reported results for TWOSIDES data instead of reproducing Decagon. However, we do not consider DrugBank data for Decagon as we do not have the additional information (e.g., side effects and target protein) in our DrugBank dataset to construct the multi-modal graph.

3.3.4 Results

3.3.4.1 Model Performances

We conduct detailed experiments on our proposed models for two different datasets with different threshold values of k-mer and ESPF. Both MLP and dot predictor-based decoder functions are employed individually for each setup to compare their performances. The overall performances are illustrated in Fig. 3.2 and Fig. 3.3. Fig. 3.2 depicts the model's performance for different ESPF frequency thresholds ranging from 5 to 25. This figure shows that it has a more significant impact on the TWOSIDES dataset, especially for the Dot decoder function than DrugBank. On TWOSIDES with MLP, it gives similar results till 25, and then it has a considerable decrease for 25. Since we get a significantly less number of substructures, it would not be enough to learn with those. For DrugBank, it gives similar results for different thresholds of ESPF. In general, frequency threshold 5 gives the best performance for TWOSIDES and DrugBank with MLP and DOT. As with the increment of the frequency threshold, the number of substructures (i.e., nodes) decreases, which could be a potential reason for performance degradation. The best performance for each dataset and decoder is recorded for a threshold value of 5.

Fig. 3.3 presents the models' performances for five different k values of k-mer ranges from 3 to 15. Similar to ESPF, the effect of the parameter on the results is higher for TWOSIDES than DrugBank. The reason for this could be that it is smaller than DrugBank, so the graph's size is affecting its results. However, DrugBank is a large dataset with enough training data to get good results, even with a small graph. Here, we can see that with

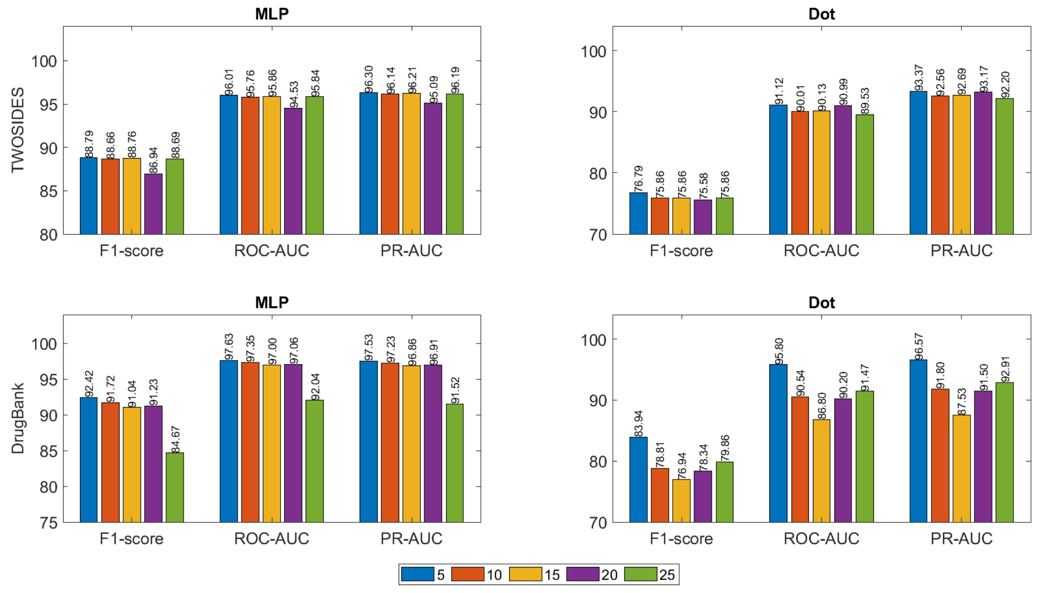


Figure 3.2 Performance comparison of models for different frequency thresholds of ESPF.

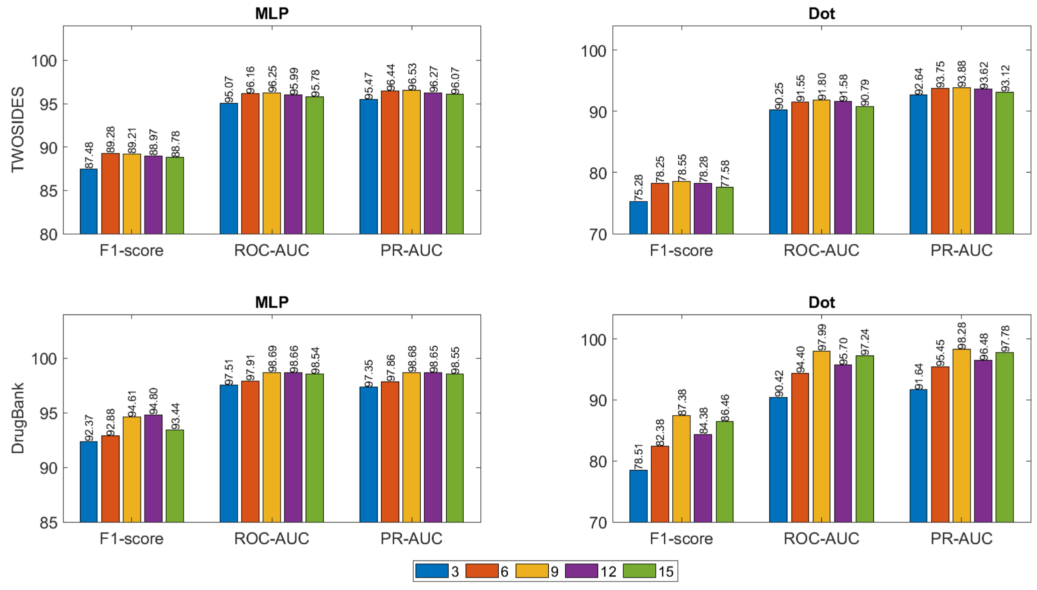


Figure 3.3 Performance comparison of models for different sizes of k-mer.

Model	Method	F1	ROC-AUC	PR-AUC
RWE on DDI Graph	DeepWalk	80.35	80.36	85.19
	Node2vec	84.50	84.52	88.33
GNN on DDI graph	GCN	85.34	85.38	88.87
	GraphSage	85.83	85.80	89.28
	GAT	82.67	82.68	86.86
GNN on SSG graph	GCN	53.85	54.04	66.94
	GraphSage	60.19	60.18	70.34
	GAT	54.25	54.37	66.85
CASTER	-	82.35	90.45	90.58
Decagon	-	-	87.20	83.20
`HyGNN`	ESPF & MLP	88.79	96.01	96.30
	ESPF & Dot	76.79	91.12	93.37
	***k*-mer & MLP**	**89.21**	**96.25**	**96.53**
	k-mer & Dot	78.55	91.80	93.88

Table 3.5 Performance comparisons of `HyGNN` with baseline models on TWOSIDES dataset.

Model	Method	F1	ROC-AUC	PR-AUC
RWE on DDI Graph	DeepWalk	73.34	73.35	80.05
	Node2vec	79.52	79.54	84.56
GNN on DDI graph	GCN	77.05	77.06	82.78
	GraphSage	80.83	80.88	85.51
	GAT	63.84	69.75	78.52
GNN on SSG graph	GCN	58.00	58.04	69.11
	GraphSage	61.10	61.15	70.64
	GAT	58.20	58.24	69.25
CASTER	-	87.36	94.27	94.20
`HyGNN`	ESPF & MLP	92.42	97.63	97.53
	ESPF & Dot	83.94	95.80	96.57
	***k*-mer & MLP**	**94.61**	**98.69**	**98.68**
	k-mer & Dot	87.38	97.99	98.28

Table 3.6 Performance comparisons of `HyGNN` with baseline models on DrugBank dataset.

the increment of the size of k-mer, the performance of the model increases, especially for TWOSIDES. As with increasing k, the number of substructures (i.e., nodes) increases which could be the reason for overall performance improvement. After some point, we will get too many substructures that could put noise into the data and decrease the model's performance. The best performance for each dataset and decoder are reported with $k = 9$ for k-mer.

3.3.4.2 Comparison with Baselines

We evaluate our models by comparing their performances with several baseline models and present the results in Table 3.5 for the TWOSIDES, and Table 3.6 for the DrugBank dataset. As we see in these tables, our models comprehensively outperform all the baseline models. More precisely, in Table 3.5 for the TWOSIDES dataset, `HyGNN` achieves at least 7% on F1, 6% on ROC-AUC, and PR-AUC better performance than other baseline models. While the best model among all the baselines, CASTER, achieves an F1 score of 82.35%, our `HyGNN` with k-mer & MLP scores 89.21% with almost 7% gain. A similar situation also happens for the other two accuracy measures: ROC-AUC and PR-AUC.

In Table 3.5, for the DDI graph, from the GNN models, GraphSage gives the best results with 85.83%, 85.80%, and 89.29% on F1, ROC-AUC, and PR-AUC, respectively. Also, from random walk-based embedding models, Node2Vec gives the best results, which is very similar to GraphSage results with 84.50% on F1, 84.52% on ROC-AUC, and 88.33% PR-AUC scores. For the SSG graph, again, GraphSage gives the best result. In our result comparison table, CASTER is the best competitor of `HyGNN`. Out of all baseline models, CASTER shows the best performance with ROC-AUC and PR-AUC of above 90%. Decagon is a multi-modal graph that also exhibits better performance than GNN on SSG.

Table 3.6 presents the performance comparison of `HyGNN` with baselines for the DrugBank dataset. As for TWOSIDES, here again, out of three different GNN models on DDI and SSG graphs, GraphSage yields the best result. Similarly, Node2Vec performs better than DeepWalk. CASTER is still the best performer among all baselines, with 87.36%, 94.27%,

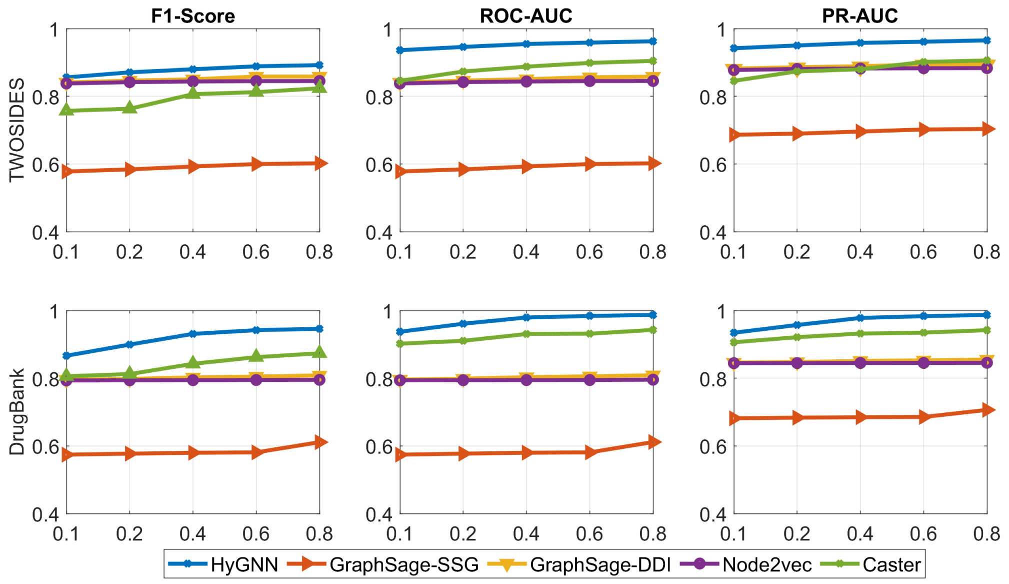


Figure 3.4 Performance comparison of models for different training sizes where x-axes represent the training percentages.

and 94.20% on F1, ROC-AUC, and PR-AUC, respectively. However, our HyGNN with k-mer & MLP significantly surpasses CASTER with 94.61% on F1, 98.69% on ROC-AUC, and 98.68% on PR-AUC. As Decagon depends on the drug and other drug-centric information, we could not experiment with it on the DrugBank dataset.

In summary, HyGNN with k-mer gives better results than ESPF. The reason for this could be that with the ESPF, we eliminate many substructures but keep just frequent ones. This may result in losing important ones that are not frequent. However, with k-mer, we get all and let the attention models in HyGNN learn which substructures are more important for DDI.

Moreover, we take the best-performing method from each baseline model, namely Node2Vec from random walk-based embedding, GraphSage from GNN on DDI, GraphSage from GNN on SSG, CASTER, and k-mer & MLP from our HyGNN models. Then, we compare their performances by changing the training sizes from 10% to 80% for both datasets. A comparison of performance is outlined in Fig 3.4. Results indicate `HyGNN` to be the best-performing model, and it still gives very good results with small training data. However, decreasing the training size affects the baseline models significantly, especially GraphSage on the SSG model. It is worthy of mention that based on our results, all graph-based models, especially different variants of GNNs, including `HyGNN` and baselines, have performed fairly well on our data.

Hypergraphs are used in a wide range of scientific fields. Hypergraphs are a natural method to illustrate shared group relationships. Through a hypergraph structure, `HyGNN` is able to capture higher-order correlations between data (i.e., triadic, tetradic, etc.). Furthermore, employing an attention mechanism makes it more robust by giving more weight to important substructures while learning representations of drugs. Though GAT has attention architecture as well, it can not discover the important edges. The main strength of our `HyGNN` is the proposed *hyperedge encoder* that has two levels of attention mechanism. At first, it aggregates the hyperedges to generate the representation of the node. While aggregating, it imposes more attention on the important hyperedges. Similarly, to generate the representation of a hyperedge, it aggregates the nodes' information with much attention to the important ones.

Moreover, `HyGNN` has a decoder function, and we learn all the parameters of the encoder and decoder simultaneously during training. From Table 3.5 and Table 3.6, we can see `HyGNN` with k-mer & MLP performs better than dot product. k-mers are k-length substrings

Drug1	Drug2	TL	Predicted DDI Score	DL
Desvenlafaxine	Paroxetine	0	0.9989	1
Probenecid	Metformin	0	0.9931	1
Fluvastatin	Metronidazole	0	0.9212	1
Loratadine	Isradipine	0	0.9703	1
Glyburide	Bosentan	0	0.9068	1
Salmeterol	Dicycloverine	0	0.9189	1
Valdecoxib	Sodium sulfate	0	0.9105	1
Lisinopril	Naratriptan	0	0.9336	1
Bexarotene	Maprotiline	0	9.9993e-10	0
Amoxapine	Econazole	0	6.8256e-09	0
Nabilone	Oxaprozin	0	4.1440e-08	0
Dexmedetomidine	Carbachol	0	1.2417e-08	0

Table 3.7 Novel DDI Predictions by `HyGNN` on the TWOSIDES Dataset

Drug1	Drug2	DL	Predicted DDI Score	TL
Hydroxychloroquine	Loratadine	0	0.9879	1
Dextromethorphan	Ofloxacin	0	0.9772	1
Midazolam	Warfarin	0	0.9884	1
Benzthiazide	Fentanyl	0	5.6989e-14	0
Labetalol	Levonorgestrel	0	9.1049e-07	0
Cefprozil	Disulfiram	0	1.0882e-11	0

Table 3.8 Novel DDI Predictions by `HyGNN` on DrugBank Dataset

included inside a biological sequence. A bigger k-mer is preferable since it ensures greater uniqueness in the base sequences that will create the string. Larger k-mer sizes aid in the elimination of repetitive substrings. Moreover, MLP predictors are well-suited for classification problems in which data is labeled. They are extremely adaptable and may be used to learn a mapping from inputs to outputs in general. Additionally, it generates superior results compared to dot predictor since it has trainable parameters that are learned throughout the training.

3.3.4.3 Case Study - Prediction and Validation of Novel DDIs

We evaluate the effectiveness of `HyGNN` model on novel DDIs prediction. We select some drug pairs from the TWOSIDES. None of these drug pairs have DDI info in TWOSIDES

Dataset	Unseen Node	F1	ROC-AUC	PR-AUC
TWOSIDES	5%	72.75	78.25	85.64
DrugBank	5%	65.23	70.84	78.04

Table 3.9 Performance of `HyGNN` for New Drugs

but have DDI info in DrugBank. Then we train our `HyGNN` using TWOSIDES and make predictions for those pairs. Following that, we collect predicted scores for those drug pairs as shown in Table 3.7. From this table, we see that for the first eight drug pairs, though the TWOSIDES label for each of these pairs is zero, we get predicted scores above 90% for each pair, which shows there is a high chance that each pair will interact between them. To further validate it, we cross-check our predicted score with DrugBank, which says each of these eight drug pairs interacts between them. Moreover, for the last four-drug pairs of Table 3.7 the predicted scores are minimal, and TWOSIDES, and DrugBank both say they don't interact. Similarly, six drug pairs are selected from DrugBank having no DDI info in DrugBank but in TWOSIDES as shown in Table 3.8, then `HyGNN` is trained using DrugBank data and validated the predicted scores by TWOSIDES.

3.3.4.4 Case Study- DDI Prediction for New Drugs

Making DDI predictions for new drugs could be more challenging than existing drugs. Since the model does not learn based on the SMILES strings of new drugs. To show the effectiveness of our model for new drugs, at first, we randomly select a 5% drug from a dataset and completely remove these drugs' information from the corresponding train set and keep those drugs' information only in the test set. These selected 5% drugs can be considered new drugs. The experimental results for both datasets with new drugs are shown in Table 3.9.

As we see in the table, our model predicts DDIs effectively for both datasets.

CHAPTER 4
SEQ-HYGAN: SEQUENCE CLASSIFICATION VIA HYPERGRAPH ATTENTION NETWORK

4.1 Introduction

In this paper, we present a novel Hypergraph Attention Network model, `Seq-HyGAN`, for sequence classification. We hypothesize that sequences sharing structural similarities tend to belong to the same classes, and sequences can be considered similar if they contain similar subsequences. To capture these structural similarities, we represent sequences in a hypergraph framework, where sequences are depicted as hyperedges connecting their respective subsequences as nodes. This allows us to create a single hypergraph encompassing all sequences in the dataset. Unlike a standard graph where each edge connects exactly two nodes, hyperedges can connect an arbitrary number of nodes, enhancing the representation of sequences and capturing complex relationships among them.

Our `Seq-HyGAN` architecture employs a three-level attention-based neural network, making it robust in capturing both local (within the sequence) and global (between sequences) information. Traditional GNNs are limited to message-passing between two nodes, but our hypergraph setting enables learning robust sequence representations through message-passing among many nodes and between nodes and hyperedges.

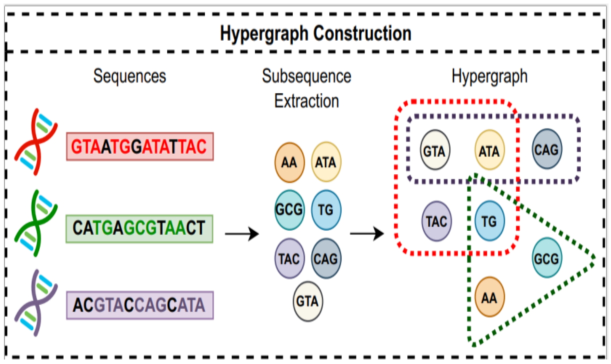
Our contributions are summarized as follows:

- **Hypergraph Construction from Sequences:** We introduce a novel hypergraph construction model where each subsequence from the sequences is represented as a node, and each sequence, composed of unique subsequences, is represented as a hyperedge. This model captures higher-order structural similarities between sequences.

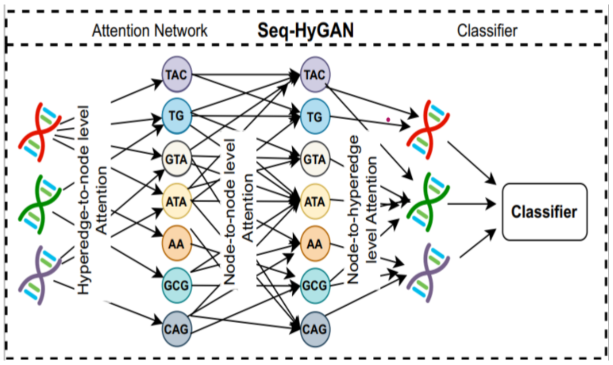
- **Hypergraph Attention Network:** We propose `Seq-HyGAN`, a hypergraph attention network model specifically designed for sequence classification. Our model learns sequence representations as hyperedges while considering both local and global context information through three levels of attention-based aggregation. At the first level, it generates node embeddings incorporating global context by aggregating hyperedge embeddings. At the second level, it refines node embeddings for each hyperedge, capturing local context by aggregating neighboring node embeddings within the same hyperedge. Finally, at the third level, it generates sequence embeddings by aggregating node embeddings from both global and local perspectives.

- **Capturing Importance via Attention:** Our model incorporates an attention mechanism to capture the relative importance of individual subsequences (nodes) within each sequence (hyperedge). This mechanism learns the varying significance of specific subsequences in contributing to the overall similarity between sequences and discerns the importance of subsequences and sequences relative to each other, capturing interdependencies at different levels of granularity.

- **Extensive Experiments:** We conduct extensive experiments on four different datasets and five classification problems, comparing `Seq-HyGAN` with state-of-the-art baseline models. The results demonstrate that our method significantly surpasses baseline mod-



(a) Hypergraph Construction


Attention Network
Seq-HyGAN
Classifier
Hyperedge-to-node level Attention
Node-to-node level Attention
Node-to-hyperedge level Attention
TAC
TG
GTA
ATA
AA
GCG
CAG
Classifier

(b) `Seq-HyGAN`

Figure 4.1 System architecture of the proposed method. The first step is hypergraph construction, where each sequence (e.g., DNA) is a hyperedge, and the (frequent) subsequences of sequences are the nodes. The second step is the Sequence Hypergraph Attention Network, namely `Seq-HyGAN`, which generates the representations of sequences while giving more attention to the important subsequences and learning the labels of the sequences.

els in various accuracy measures.

4.2 Methodology

4.2.1 Preliminaries and Settings

Sequence classification is the problem of predicting the class of sequences. Our motivation in this work is that patterns as subsequences are important features of sequences, and if two sequences share many patterns, they have a higher similarity. Also, it is assumed that similar sequences will have the same class labels. To define the higher-order pattern-based similarity between sequences, we construct a hypergraph from the sequence data.

After hypergraph creation, to accomplish the sequence classification task, we propose an attention-based hypergraph neural network model consisting of a novel three-level attention mechanism that learns the importance of the subsequences (nodes) and, thus, the representation of the sequences (hyperedges). We train the whole model in a semi-supervised fashion. Our proposed model has two steps: (1) Hypergraph construction from the sequence data, (2) Sequence classification using attention-based hypergraph neural networks. Figure 4.1 shows the overall model architecture.

4.2.2 Sequence Hypergraph Construction

In order to capture the similarity between sequences, we define the relationship between sequences based on common subsequences within each sequence. We represent this relationship as a hypergraph that captures the higher-order similarity of the sequences. First, we decompose the sequences into a set of subsequences as the important patterns of the sequences. Then, we represent this set of subsequences as the nodes of the hypergraph, and each sequence, including a set of subsequences, is a hyperedge. Each hyperedge may connect with other hyperedges through some shared nodes as subsequences. Thus, this constructed hypergraph defines a higher-level connection of sequences and subsequences and helps to capture the complex similarities between the sequences. Moreover, this hypergraph setting ensures a better robust representation of sequences with a GNN model having a message-passing mechanism not only limited to two nodes but rather between the arbitrary number of nodes and also between edges through nodes. The steps of hypergraph construction are shown in Algorithm 4.

Different algorithms can be used to generate the subsequences, such as ESPF [118], k-mer [119], strobemers [120]. In this paper, we exploit ESPF, and k-mer to generate subsequences and examine their effects on the final results. While k-mer uses all the extracted subsequences for a certain k value, ESPF only selects the most frequent subsequences from a list of candidate subsequences for a certain threshold.

Algorithm 4: Sequence Hypergraph Construction

```
Input: Sequences
Output: Hypergraph incident matrix: $A$
Subsequence_list¡- $Sequence\_Decomposition(Sequences)$;
 /* Sequence_Decomposition() could be ESPF or $k$-mer that decomposes
 sequences into moderated size subsequences.  */
for each subsequence in Subsequence_list do
    if subsequence is in Sequence_dictionary[sequence] then
        $A[i, j] = 1$ ;          /* i,j is the id of subsequence and sequence,
         respectively.  */
    end
end
Output: Hypergraph incident matrix, $A$
```

ESPF: ESPF stands for Explainable Substructure Partition Fingerprint. As for sub-word mining in the natural language processing domain, ESPF decomposes sequential inputs into a vocabulary list of interpretable moderate-sized subsequences. ESPF considers that a specific sequence property is mainly led by only a limited number of subsequences known as functional groups. Given a database, S of sequences as input, ESPF generates a vocabulary list of subsequences as frequent reoccurring customized size subsequences. Starting with tokens as the initial set, it adds subsequences having a frequency above a threshold in S to the vocabulary list. Subsequences appear in this vocabulary list in order of most frequent to least frequent. In our hypergraph, we use this vocabulary list as nodes and break down any sequence into a series of frequent subsequences relating to those. For any given sequence as input, we split it in order of frequency, starting from the highest frequency one. An example of splitting a DNA sequence is as follows.

`CTGAAAGCAACAGTGAGACGATGAGACCGACGATCCCAGGAGG`

$$\Downarrow$$

`CTGAAAG` `CAACAG` `TGAGA` `CGA` `TGAGA` `CCGACGA` `TCCCAG` `GAGG`

k-mer: k-mer is an effective tool widely used in biological sequence data analysis (e.g., sequence matching). It splits sequential inputs into a list of overlapping subsequence strings of length k. To generate all k-mers from an input string, it starts with the first k characters and then moves by just one character to get the next subsequence, and so on. If t is the length of a sequence, there are $t - k + 1$ numbers of k-mers and T^k total possible number of k-mers, where T is the number of monomers. k-mers can be considered as the words of a sentence. Like words, they help to attain the semantic features from a sequence. For example, for a sequence `ATGT`, monomers: {A, T, and G}, 2-mers: {AT, TG, GT}, 3-mers: {ATG, TGT}.

4.2.3 Sequence Hypergraph Attention Network

To classify sequences, it is essential to generate feature vectors that can effectively embed the structural information. Since in our hypergraph model, we represent each sequence as a hyperedge; we need to learn hyperedge representation. Regular GNN models that generate the embedding of nodes do not work on our hypergraph. Therefore, we propose a novel Sequence Hypergraph Attention Network model, namely `Seq-HyGAN`.

Given the f dimensional hyperedge feature matrix, $X \in R^{|\mathcal{E}| \times f}$ and incidence matrix $A \in R^{|\mathcal{V}| \times |\mathcal{E}|}$, `Seq-HyGAN` generates a hyperedge feature vector of f' dimension via learning a function F. Then, it predicts labels for sequences using generated feature vectors.

Our proposed model leverages memory-efficient self-attention mechanisms to capture high-order relationships in the data while preserving both local and global context information. It comprises a three-level attention network: hyperedge-to-node, node-to-node, and node-to-hyperedge levels. At the hyperedge-to-node level, attention is utilized to aggregate hyperedge information and generate node representations that encapsulate global context. The node-to-node level attention refines the node representations by aggregating information from neighboring nodes within the same hyperedge, capturing local context. Lastly, the node-to-hyperedge level attention aggregates node representations from both local and global context perspectives to generate hyperedge representations with attention. We define tree attention layers in general as follows.

$$p_i^l = \mathbf{AG_{E-V}}^l(p_i^{l-1}, n_j^{l-1} | \forall e_j \in \mathcal{E}_i), \tag{4.1}$$

$$m_{i,j}^l = \mathbf{AG_{V-V}}^l(p_i^l, p_y^l | \forall v_y \in e_j), \tag{4.2}$$

$$n_j^l = \mathbf{AG_{V-E}}^l(n_j^{l-1}, p_i^l, m_{i,j}^l | \forall v_i \in e_j) \tag{4.3}$$

where $\mathbf{AG_{E\text{-}V}}$ (Hyperedge-to-Node) aggregates the information n_j of all hyperedges e_j to generate the l-th layer representation p_i^l of node v_i and $\mathcal{E}_i$ is the set of hyperedges that node v_i belongs to. $\mathbf{AG_{V\text{-}V}}$(Node-to-Node) generate the l-th layer representation $m_{i,j}^l$ of node v_i for a specific hyperedge e_j by aggregating all the nodes v_y present in e_j. Finally, $\mathbf{AG_{V\text{-}E}}$(Node-to-Hyperedge) aggregates the information of all nodes v_i that belongs to hyperedge e_j to generate the l-th layer representation n_j^l of e_j.

Hyperedge-to-node level attention: As our first layer, we learn the representation of nodes via aggregating information from hyperedges to capture the global context in the hypergraph. Although a node may belong to different hyperedges, all hyperedges may not be equally important for that node. To learn the importance of hyperedges for each node and incorporate them into the representation of nodes, we design a self-attention mechanism. While aggregating hyperedge representations for a node, this attention mechanism ensures more weight to important hyperedges than others. With the attention mechanism, the l-th layer node feature p_i^l of node v_i is defined as

$$p_i^l = \alpha\left(\sum_{e_j \in \mathcal{E}_i} \Gamma_{ij} W_1 n_j^{l-1}\right) \tag{4.4}$$

where α is a nonlinear activation function, W_1 is a trainable weight matrix, and Γ_{ij} is the attention coefficient of hyperedge e_j on node v_i defined as

$$\Gamma_{ij} = \frac{\exp(\mathbf{e_j})}{\sum_{e_k \in E_i} \exp(\mathbf{e_k})} \tag{4.5}$$

$$\mathbf{e_j} = \beta(W_2 n_j^{l-1} * W_3 p_i^{l-1}) \tag{4.6}$$

where $\mathcal{E}_i$ is the set of hyperedges v_i is connected with, β is a LeakyReLU activation function and $*$ is the element-wise multiplication and W_2 and W_3 are trainable weights.

Node-to-node level attention:

The hyperedge-to-node level attention captures global context information while generating node representations. However, while a subsequence is common in the different sequences, they may also have different roles and importance in each sequence. Therefore, it is crucial to capture the local information of nodes specific to hyperedges. Moreover, we need to incorporate the individual contributions of adjacent local subsequences into the representation of a specific subsequence. Furthermore, retaining the positional information of the subsequences in a sequence is essential for the accurate analysis of sequence data. To address these, we introduce a node-to-node attention layer that passes information between nodes in a hyperedge. It learns the importance of subsequences for each other within the same sequence and also incorporates a position encoder that assigns a unique position to each subsequence. To get the position information, we adopt a simple positional encoder inspired by the transformer model [41]. This enables us to preserve local and spatial information of a subsequence within a specific sequence. Using the attention mechanism, the l-th layer node feature $m_{i,j}^{l}$ of node v_i belonging to hyperedge e_j is defined as

$$m_{i,j}^{l} = \alpha \left(\sum_{v_y \in e_j} \Phi_{iy} W_4 \bar{p_y}^{\,l} \right) \tag{4.7}$$

$$\bar{p}_y = p_y + \mathrm{PE}(pos_{v_y}) \tag{4.8}$$

$$\mathrm{PE}(pos, 2x) = \sin(pos/10000^{2x/d}) \tag{4.9}$$

$$\mathrm{PE}(pos, 2x+1) = \cos(pos/10000^{2x/d}) \tag{4.10}$$

Where W_4 is a trainable weight, Φ_{iy} is the attention coefficient of neighbor node v_y on node v_i. PE represents the positional encoding function, pos_{v_y} represents the original positional index of v_y in the sequence, $\mathrm{PE}(pos, x)$ refers to the x-th dimension of the positional encoding of the word at position *pos* in the sequence, and d denotes the dimension of the positional encoding. The attention coefficient Φ_{iy} is defined as

$$\Phi_{iy} = \frac{\exp(\mathbf{q_y})}{\sum_{q_k \in e_j} \exp(\mathbf{q_k})} \tag{4.11}$$

$$\mathbf{q_y} = \beta(W_5 \bar{p_y}^l * W_6 \bar{p_i}^l) \tag{4.12}$$

where e_j is the hyperedge node v_i belongs, W_5 and W_6 are trainable weights.

Node-to-hyperedge level attention: Hyperedge is degree-free that consists of an arbitrary number of nodes. However, the contribution of nodes in hyperedge construction may not be the same. To highlight the important nodes for each hyperedge, we employ an attention mechanism. This attention aggregates node representations and assigns higher weights to crucial ones. Moreover, during the aggregation process, it considers the representations of the nodes from both local and global contexts, allowing for a comprehensive understanding of their significance within the hypergraph. With the attention mechanism, the l-th layer hyperedge feature n_j^l of hyperedge e_j is defined as

$$n_j^l = \alpha \left(\sum_{v_i \in e_j} \Delta_{ji} W_7 (m_{i,j}^l || p_i^l) \right) \tag{4.13}$$

where W_7 is a trainable weight, $||$ is the concatenation operation, and Δ_{ji} is the attention coefficient of node v_i in the hyperedge e_j defined as

$$\Delta_{ji} = \frac{\exp(\mathbf{v_i})}{\sum_{v_k \in e_j} \exp(\mathbf{v_k})} \tag{4.14}$$

$$\mathbf{v_i} = \beta \left(W_8 (m_{i,j}^l || p_i^l) * W_9 n_j^{l-1}) \right) \tag{4.15}$$

where v_k is the node that belongs to hyperedge e_j, and W_8, W_9 are trainable weights.

`Seq-HyGAN` generates the hyperedge representations by employing this three-level of attention. Finally, we linearly project the output of `Seq-HyGAN` with a trainable weight matrix to generate a C dimensional output for each hyperedge as $Z = nW_c^T$, where C is the number of classes, n is the output of the `Seq-HyGAN`, and W_c is a trainable weight.

Training: We train our entire model using a cross-entropy loss function defined as

$$L = -\sum_{i=1}^{N} \sum_{c=1}^{C} w_c \log \frac{\exp(Z_{i,c})}{\sum_{j=1}^{C} \exp(Z_{i,j})} y_{i,c} \tag{4.16}$$

where y is the target, w is the weight, C is the number of classes, and N is the number of samples.

4.2.4 Complexity

`Seq-HyGAN` is an efficient model that can be parallelized across the edges and the nodes [121]. Given the f dimensional initial feature of a sequence, it exploits a three-level attention network and generates f' dimensional embedding vector for the sequence. Thus, the time complexity of `Seq-HyGAN` can be expressed in terms of the cumulative complexity of the attention networks. From equation 4.4, we can formulate the time complexity for the hyperedge-to-node level attention as: $O(|\mathcal{E}|ff' + |\mathcal{V}|\kappa f')$, where κ is the average degree of nodes. And in the node-to-node level attention, the time complexity is: $O(|\mathcal{E}|(\chi ff' + \chi^2 f'))$, where χ is the average degree of hyperedges. Similarly, we can formulate the time complexity in the node-to-hyperedge level attention as: $O(|\mathcal{V}|ff' + |\mathcal{E}|\chi f')$.

4.3 Experiment

In this section, we perform extensive experiments on four different datasets and five different research problems to evaluate the proposed `Seq-HyGAN` model. We compute three different accuracy metrics, Precision (P), Recall (R), and F1-score (F1), to analyze and compare our proposed model with the state-of-the-art baseline models. This section starts with a description of our datasets, parameter settings, and baselines, and then we present our experimental results.

4.3.1 Dataset

We evaluate the performance of our model using four different sequence datasets. They are (1) **Human DNA** sequence, (2) **Anti-cancer peptides**, (3) **Cov-S-Protein-Seq** and (4) **Bach choral** harmony. All these datasets are publicly available online.

1. The **Human DNA** (HD) sequence dataset consists of 4,380 DNA sequences [124]. Each DNA sequence corresponds to a specific gene family (class), with a total of seven families.

Our objective is to predict the gene family based on the coding sequence of the DNA.

2. The **Anti-cancer peptides** (ACPs) are short bioactive peptides [125]. ACPs are found to interact with vital proteins to inhibit angiogenesis and recruit immune cells to kill cancer cells, such as HNP-110 [106]. These unique advantages make ACPs the most promising anti-cancer candidate [126]. The ACP dataset contains 949 one-letter amino-acid

sequences representing peptides and their four different anti-cancer activities (i.e., very active, moderately active, experimental inactive, virtual inactive) on breast and lung cancer cell lines [127]. Given the amino-acid sequence, our goal is to predict the anti-cancer activities.

3. The **Cov-S-Protein-Seq** (CPS) dataset consists of 1,238 spike protein sequences from 67 different coronavirus (CoV) species, including SARS-CoV-2 responsible for the COVID-19 pandemic [128, 129]. The dataset provides information on the CoV species (CVS) and their host species (CHS) [130]. The CoV species are grouped into seven categories, and the host species are grouped into six categories. The goal is to predict the CoV species and host species based on the spike protein sequences.

4. Music is sequences of sounding events. Each event has a specific chord label. The **Bach choral** (BC) harmony dataset consists of 60 chorales containing a total of 5,665 events [127]. Each event is labeled with one of 101 chord labels and described by 14 features. The goal is to predict the chord label based on this information. For the experiment, the five most frequent chord labels out of the 101 chord labels are selected.

4.3.2 Parameter Settings

We extract subsequences from given sequence datasets using ESPF and k-mer separately to create our hypergraph. With a low-frequency threshold, ESPF produces more subsequences, and all of them may not be important. But with a large-frequency threshold, it produces less number of subsequences, and there might be a missing of some vital subsequences. We choose five different frequency thresholds from 5 to 25 and examine the impact of threshold value on hypergraph learning. Similarly, in k-mer, typically with the increment of k-mer length (i.e.,

ESPF	HD	BC	ACP	CPS
	$\lvert N\rvert$	$\lvert N\rvert$	$\lvert N\rvert$	$\lvert N\rvert$
5	25207	446	382	10776
10	15774	347	225	7987
15	11166	287	166	6769
20	8871	253	138	5971
25	7483	198	110	5477

k-mer	HD	BC	ACP	CPS
	$\lvert N\rvert$	$\lvert N\rvert$	$\lvert N\rvert$	$\lvert N\rvert$
5	1247	3220	10301	99794
10	602,855	14462	6799	157,399
15	1,449,240	16163	3103	193,544
20	1,462,963	17909	-	223,073
25	1,467,256	18752	-	248,938

Table 4.1 Number of Nodes (N) in the hypergraph based on frequency threshold of ESPF and k value of k-mer

k value), the number of subsequences also increases. We choose five different k values from 5 to 25 and examine their impact on hypergraph learning. However, as Anticancer peptide sequences are too small, we just choose k from 5 to 15. The number of nodes for different threshold values of ESPF and k value of k-mer is given in Table 4.1 for each dataset.

We perform a random split of our datasets, dividing them into 80% for training, 10% for validation, and 10% for testing. This splitting process is repeated five times, and the average accuracy metrics are calculated and reported in the results section. To find the optimal hyperparameters, a grid search method is used on the validation set. The optimal learning rate is determined to be 0.001, and the optimal dropout rate is found to be 0.3 to prevent overfitting.

We utilize a single-layer `Seq-HyGAN` having a three-level of attention network. one-hot coding is used as an initial feature of the sequences. A LeakyReLU activation function is used on the attention networks side. The model is trained with 1000 epochs and optimized using Adam optimizer. An early stop is used if the validation accuracy does not change for

100 consecutive epochs.

The ML classifiers in subsection: 4.3.3 are taken from sci-kit learn [131]. For logistic regression (LR), we set the inverse of regularization strength, C=2. A linear kernel with polynomial degree 3 is used in the support vector machine (SVM). Default parameters are used for the decision tree (DT) classifier. The DL models are implemented from Keras layers [132]. In the DL models, RCNN and BiLSTM, we use relu and softmax activation functions in the hidden dense layers and dense output layer, respectively. The models are optimized using Adam optimizer, and dropout layers of 0.3 are used. For node2vec, the walk length, number of walks, and window size are set to 80, 15, and 15, respectively, and for graph2vec, we use the default parameters following the source. We follow DGL [133] to implement graph attention network (GAT). For DNA-GCN and hypergraph neural networks (HGNN, HyperGAT), we use the same hyper-parameters as mentioned in the source papers.

4.3.3 Baselines

We evaluate our model by comparing its performance with different state-of-the-art models. Based on the data representation style and methodology, we categorize the baseline models into groups below.

1. Machine Learning We utilize CountVectorizer to generate the input features and employ LR, SVM, and DT classifiers. *2. Deep Learning* We use two different hybrid DL models, recurrent convolutional neural networks (RCNN) and bidirectional long short-term memory (BiLSTM), as baselines. *3. Node2vec* We represent each sequence in a graph setting by following a classic method called the De-Bruijn graph as explained in section 2.3.2.

After constructing the graph, we apply Node2vec [123], which is a random walk-based graph embedding. Node2vec generates the embedding of nodes. To obtain the graph-level representation, we average the embedding of nodes of that graph. Finally, the generated embedding is fed as a feature to the ML classifier for sequence classification. *4. Graph2vec* In the same De-Bruijn graph, we apply the Graph2vec [134] method to generate graph embeddings. Then the graph embeddings are fed into the ML classifiers for sequence classification. *5. Graph Neural Network* Here, we apply graph attention network (GAT) [121] on De-Bruijn graphs to learn the node embedding. Then we get the graph-level embedding using an average pooling-based readout function. Moreover, we follow DNA-GCN [31] to construct a heterogeneous graph from the entire corpus and the extracted subsequences. This graph has two types of nodes: sequence node and subsequence node. After constructing the graph, we apply a two-layer GCN. *6. Hypergraph Neural Network (HNN)* We further compare our model performances with two state-of-the-art hypergraph neural network models: HGNN [7] and HyperGAT [10]. HGNN generates the representation of nodes by aggregating hyperedges. First, we apply HGNN to our hypergraphs and get the node (i.e., subsequence) representations, and then we combine the node representations to get the hyperedge representations. Finally, the hyperedge representations are passed through a classifier. HyperGAT is an attention-based hypergraph neural network that is presented for document classification problems. We apply HyperGAT to our sequence hypergraphs and generate the embeddings of subsequences, and then we apply a mean-pooling layer to get the sequence embedding. Then, the representation is passed through a classifier. In both HGNN and HyperGAT, we use one-hot coding

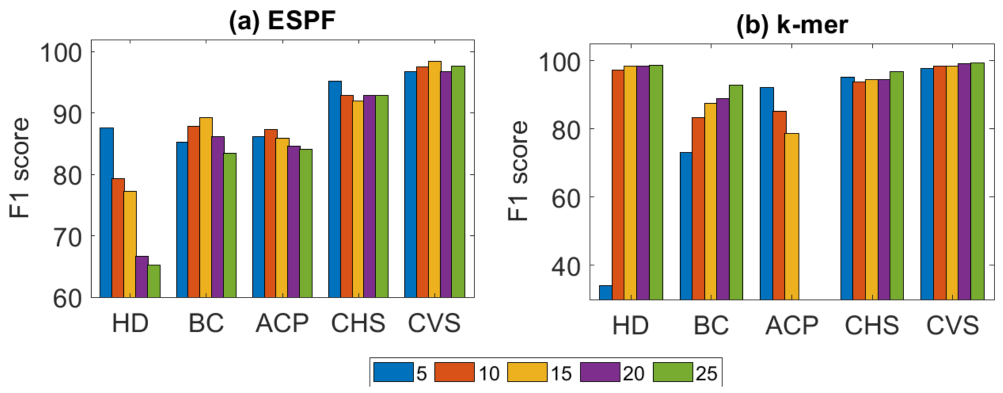


Figure 4.2 Performance comparison of the proposed model with different thresholds of (a) ESPF and (b) k-mer for different datasets.

of nodes as initial features.

4.3.4 Results

4.3.4.1 Model Performance

We assess the performance of our proposed model by performing extensive experiments on four different datasets for five different problems. For each experiment, we select different threshold values of ESPF and k-mer, and we present the overall performance of our proposed models in terms of the F1-score in Figure 4.2.

In Figure 4.2 (a), we depict the performance of our models with a changing frequency threshold of ESPF from 5 to 25. As we can see from this figure, with the increase of ESPF frequency threshold, model performance degrades generally. Especially it shows that the ESPF frequency threshold has a comparatively more significant impact on the Human DNA dataset than others. With the change of frequency threshold from 5 to 25, the F1 score of the Human DNA dataset has dropped by almost 25%. Since with the increase of frequency threshold, we get a smaller number of subsequences (nodes), and it might not be enough to learn hyperedges with those smaller numbers of nodes (refer Table 4.1). Generally, frequency threshold 5 yields the best performance for Human DNA and CoV-S-Protein-Seq (for Host species classification) datasets. We get the best performance for Bach choral, Anticancer peptides, and CoV-S-Protein-Seq (for CoV species classification) datasets with frequency thresholds of 15, 10, and 15, respectively.

Figure 4.2 (b) presents the proposed models' performances for k-mer ranges from 5 to 25. As of ESPF, the effect of parameter k on the results is the most for Human DNA than other datasets. When the k value is 5, the F1-score of Human DNA is 33.98%, and for the k value of 25, it is increased by about 190% to 98.83%. The next highest changes are noticed for the Bach choral dataset. With the increase of threshold k from 5 to 25, its F1-score is increased by about 27%. The value of k has comparatively less impact on the results of Cov-S-Protein-Seq datasets. However, interestingly, with the increase of k, the results of the Anticancer peptides dataset decrease. The reasons for these scenarios could be explained by Table 4.1. In this table, we can see that with the increase in k, the number of nodes in the Human DNA dataset has markedly increased from 1,247 to 1,467,256. This vast number of nodes might capture better information and thus improve the overall performance. On the other hand, for Anticancer peptides, with the increase of k, the number of nodes decreases, hence the graph size, which might degrade the performance.

Model	Method	Human DNA			Bach choral			Anticancer pept.		
		P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
ML	LR	92.82	90.64	90.84	88.09	76.68	78.52	77.00	83.16	77.67
	SVM	90.09	85.39	85.83	89.30	80.27	82.08	83.10	86.32	83.27
	DT	92.87	80.37	83.68	86.49	70.85	73.92	78.28	85.32	81.35
DL	RCNN	68.84	37.90	27.86	76.83	71.30	68.23	69.62	80.00	73.34
	BiLSTM	77.80	39.27	35.18	73.93	69.96	66.32	65.70	81.05	72.57
Node2vec	LR	36.09	32.19	22.89	16.11	20.17	18.14	71.32	82.11	75.11
	SVM	10.32	30.82	14.52	14.14	20.18	16.17	66.39	81.05	72.99
	DT	18.04	18.26	18.13	23.03	22.87	22.89	68.75	64.21	66.40
Graph2vec	LR	21.07	26.94	23.63	23.34	19.73	17.81	66.39	81.05	72.99
	SVM	10.32	30.82	14.52	13.09	15.75	16.50	71.32	82.11	75.11
	DT	19.41	19.63	19.39	25.60	25.56	25.48	77.93	73.68	75.54
GNN	DNA-GCN	96.46	96.28	96.36	85.54	85.24	85.27	83.25	83.53	83.82
	GAT	30.06	42.14	36.01	24.75	29.19	31.12	79.23	87.44	79.67
HNN	HGNN	87.03	86.82	87.12	86.12	86.89	86.93	83.82	85.42	83.97
	HyperGAT	85.13	85.33	84.11	88.09	87.44	87.45	85.33	88.42	86.68
`Seq-HyGAN`	ESPF	88.77	87.89	87.78	89.93	89.72	89.88	91.98	86.75	87.65
	***k*-mer**	**98.91**	**98.88**	**98.83**	**93.78**	**93.10**	**93.18**	**93.36**	**91.72**	**92.33**

Table 4.2 Performance comparisons of models for Human DNA, Bach choral and Anticancer datasets

Method	Model	Host species			CoV species		
		P	**R**	**F1**	**P**	**R**	**F1**
ML	LR	92.64	91.94	91.48	96.04	95.16	95.21
	SVM	94.20	93.55	93.42	96.60	95.97	96.02
	DT	92.25	91.13	91.25	95.60	94.97	95.02
DL	RCNN	83.22	79.03	76.82	65.53	62.90	57.47
	BiLSTM	70.61	66.94	61.56	66.66	72.58	66.92
Node2vec	LR	26.28	29.03	19.29	12.34	20.16	14.92
	SVM	23.07	24.19	23.27	22.42	25.00	23.54
	DT	20.00	21.77	20.80	23.47	23.39	23.33
Graph2vec	LR	23.74	25.00	23.98	25.92	28.23	26.67
	SVM	17.22	27.42	17.81	17.16	22.58	16.59
	DT	22.68	22.58	22.50	22.80	22.58	22.47
GNN	DNA-GCN	90.91	90.18	91.11	94.34	94.57	94.13
	GAT	22.36	33.23	25.22	24.10	29.65	26.19
HNN	HGNN	91.52	91.91	91.60	94.62	94.42	94.63
	HyperGAT	93.44	93.55	93.14	95.52	95.35	95.45
`Seq-HyGAN`	ESPF	95.78	95.66	95.49	98.89	98.78	98.72
	***k*-mer**	**97.83**	**96.13**	**97.01**	**99.56**	**99.29**	**99.45**

Table 4.3 Performance comparisons of models on Cov-S-Protein-Seq dataset for Host and CoV species prediction

4.3.4.2 Comparative Analysis with Baselines

We compare our model results with several state-of-the-art baseline models for each dataset.

We consider the thresholds for ESPF and k for k-mer that give the best result for our models for each dataset; as an example, for the Human DNA dataset, we chose $k = 25$ for k-mer that gives the highest score for this data. The experimental results of all models for Human DNA, Bach choral, and Anticancer peptides (Anticancer pept.) datasets are shown in Table 4.2, and all model results for Cov-S-Protein-Seq dataset for both host species and CoV species are shown in Table 4.3. In these tables, we can see our models surpass all the baselines thoroughly for all the datasets. More specifically, in Table 4.2 for the Human DNA dataset, our `Seq-HyGAN` with k-mer gives the best performance with 98.91%, 98.88%, and 98.83% on precision (P), recall (R), and F1-score (F1). The next best performer is DNA-GCN, which achieves an F1-score of 96.36%, more than 2.5% lower than our model and similar to other accuracy measures. The performances of ML models are also very promising and competitive with our models. All the ML models achieve an F1-score above 80%.

Eventually, for all the datasets, the hypergraph-based model gives the best performance. Specifically, in almost every case, HyperGAT serves as the superior baseline, while HGNN performs as the second-best baseline. The reason for their success lies in their ability to capture higher-order, intricate relationships within a hypergraph structure. Additionally, HyperGAT utilizes attention networks to enhance sequence representation learning, which is superior to HGNN's approach. It is worth mentioning that we apply these models to our hypergraphs, and our experiments demonstrate the effectiveness of representing sequences as hypergraphs.

The overall performance of DNA-GCN is also very competitive with hypergraph-based approaches. DNA-GCN is based on a heterogenous graph that has both sequence and subsequence nodes. It allows information to be passed between the subsequences and also between sequence and subsequence. Moreover, though it does not have any direct connection between sequence nodes, it employs a two-layer GCN that allows the information to be passed between the sequences too. This whole architecture helps it learn a robust representation of sequences. Out of all ML models, generally, SVM generates better output for all the datasets. However, ML models cannot learn features automatically and are limited to external features. In both tables, for all the datasets, the performances of DL models fail to cross the ML classifiers. For example, in the Bach choral dataset, the F1-score of `Seq-HyGAN` with k-mer is above 93%, and for ML with SVM classifier, it is above 82%. However, the F1-score for RCNN and BiLSTM are 68.23% and 66.32%, respectively. A similar scenario in all other datasets. A possible reason behind the poor performance of DL models could be the size of the data corpus. We know DL models are called data-hungry models. Their performances largely rely on the availability of a bulk amount of label data. However, all our datasets are small.

From Table 4.2, and 4.3, we can observe that the performance of Node2vec, Graph2vec, and GAT on the Anticancer peptides dataset is convincing, but their performances on other datasets are abysmal. The network structure of these datasets may be a contributing factor to this discrepancy. To further investigate, we calculate the average network density of each dataset by computing the mean density of its corresponding graphs. For each dataset, the best k-mer is chosen based on the performances of `Seq-HyGAN` models on that dataset. We find that the Anticancer peptides dataset exhibits the highest average network density of

0.3868, while the Human DNA, Bach Choral, and CoV-S-Protein-Seq dataset graphs have lower densities of 0.0100, 0.2702, and 0.0031, respectively. This disparity in network density could explain the subpar performance of Node2vec, Graph2vec, and GAT on these datasets.

In brief, `Seq-HyGAN` with k-mer delivers better performances than `Seq-HyGAN` with ESPF. This may be because ESPF assumes frequent subsequences are the only important ones and eliminates many infrequent subsequences. However, some infrequent subsequences may also be important. Thus, using ESPF, we may seldom lose some infrequent important ones. On the contrary, k-mer does not lose any subsequences; rather, it fetches all and lets the attention model discover the critical ones. In general, a larger k-mer is preferable since it provides greater uniqueness and helps to eliminate the repetitive substrings. Moreover, we choose the best-performing method from each baseline model for each dataset; for example, in the case of the Human DNA dataset, we choose LR from the ML models, BiLSTM from the DL models, LR from Graph2vec, DNA-GCN from GNN, HyperGAT from HNN and 25-mer from our `Seq-HyGAN` models. Then, we compare our models' performances with the baselines by varying the training data sizes from 10% to 80%. A comparison of performance in terms of the F1-score is shown in Fig 4.3. Results indicate `Seq-HyGAN` to be the best-performing model, and it still delivers very good results with small training data. However, decreasing the training size affects some baseline models significantly.

The hypergraph's innate ability to capture complex higher-order relationships has made it an effective model for many scientific studies. `Seq-HyGAN` leverages a hypergraph structure and captures higher-order intricate relations of subsequences within a sequence and

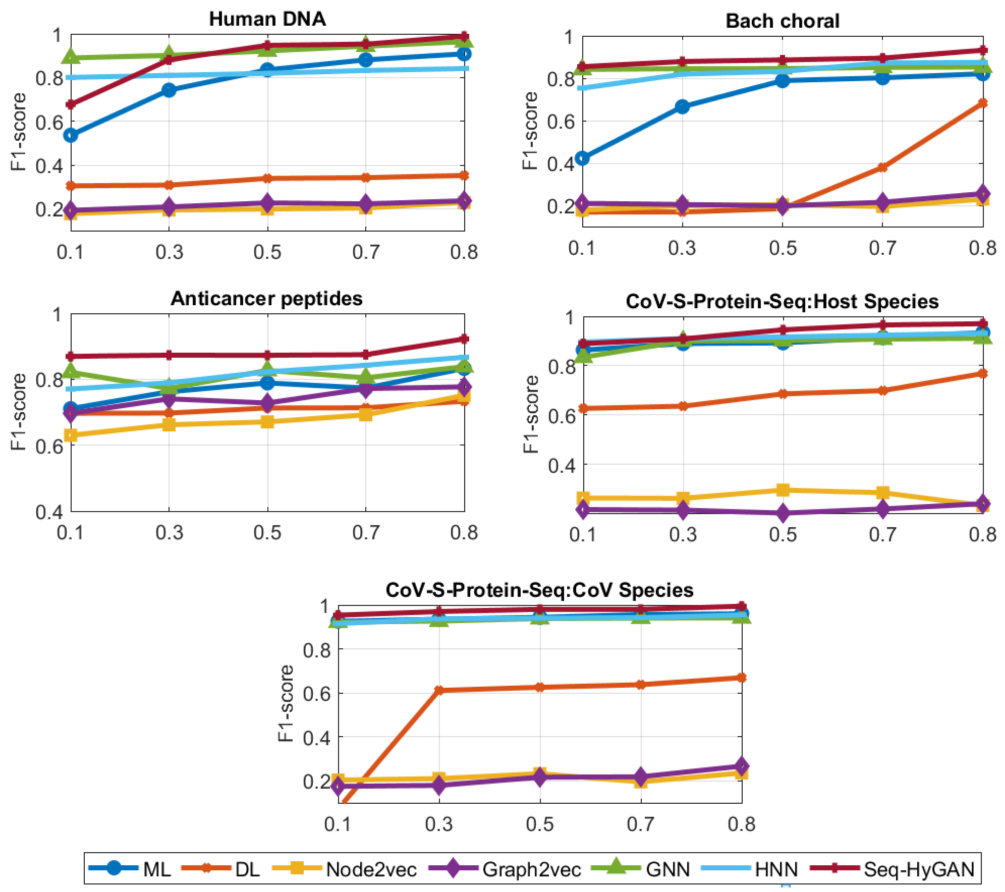


Figure 4.3 Performance comparison of models for different training sizes

between the sequences. Furthermore, it generates a much more robust representation of sequence by utilizing an attention mechanism that discovers the important subsequences of that sequence. While GAT [121] also uses an attention mechanism, it is limited to learning important neighbors of a node and cannot learn significant edges. Additionally, GAT is unsuitable for complex networks with triadic or tetradic relations. The key strength of our proposed model, `Seq-HyGAN`, lies in its three-level attention mechanism, which effectively captures both local and global information. This mechanism allows for the generation of node representations by aggregating information from connected hyperedges (global information)

Dataset	**# of nodes**	**# of edges**
	$\|N\|$	$\|E\|$
Human DNA	5,422,447	5,418,151
Bach choral	34111	31890
Anticancer peptides	11939	11058
Cov-S-Protein-Seq	1,576,019	1,574,782

Table 4.4 Number of Nodes (N) and Edges (E) in De-Bruijn Graphs

and neighboring nodes (local information) within the same hyperedge, with a specific emphasis on important ones. Likewise, it enables the generation of hyperedge representations by aggregating member nodes, with a particular focus on critical ones.

4.3.4.3 Space Analysis

Our models demonstrate efficient memory usage by creating only one hypergraph per dataset. Regardless of the chosen thresholds for ESPF and k for k-mer, the number of hyperedges remains consistent across the hypergraphs. In contrast, the De-Bruijn method constructs a separate graph for each sequence, resulting in a significant number of nodes and edges. For instance, in Table 4.1, we can see that the hypergraph constructed from the Human DNA dataset with a k-mer value of 25 contains 1,467,256 nodes and where the number of hyperedges is the same as the number of sequences in that dataset as mentioned in 4.3.1 which is 4380. However, in Table 4.4, we can see that the De-Bruijn graph constructed from the same dataset with the same k-mer value comprises 5,422,447 nodes and 5,418,151 edges. Similar trends are observed in other datasets as well.

Dataset	**Line graph**		`Seq-HyGAN`	
	GCN	**GAT**	**w/o attn**	**w/ attn**
Human DNA	37.08	39.19	94.13	98.83
Bach coral	75.41	77.65	91.21	93.18
Anticancer peptides	83.52	86.86	89.88	92.33
Host species	72.70	75.31	90.16	97.01
CoV species	80.83	86.50	95.77	99.45

Table 4.5 F1-score scores for different variants of the model

4.3.4.4 Case Study - Impact of hypergraph structure

In contrast to standard graphs, hypergraphs offer the ability to capture higher-order complex relationships that are not easily represented by standard graphs. To demonstrate this capability, we conduct a comparison between our hypergraph-based `Seq-HyGAN` model and standard graphs. To facilitate this comparison, we construct line graphs from the same datasets, where each sequence is represented as a node, and nodes are connected if they share a certain number (S) of common subsequences (with $S = 2$ in our case). Subsequently, we apply GCN and GAT independently to learn node representations and classify sequences. The performance results are presented in Table 4.5. The results clearly indicate that our hypergraph-based `Seq-HyGAN` models outperform line graph GCN and GAT models in terms of the F1-score.

4.3.4.5 Case Study - Impact of Attention Network

In this research paper, we aim to investigate the impact of the attention network in the proposed `Seq-HyGAN` model on classification performance. To achieve this, we train the model separately with and without the attention network on all datasets and classification problems. The corresponding F1-scores for the test datasets were recorded and are presented in Table

4.5. The results show that the proposed `Seq-HyGAN` model with the attention network (w/ attn) outperforms the model without the attention network (w/o attn). Specifically, for CoV-S-Protein-Seq: Host species, the F1-score improved from 90.16% without the attention network to 97.01% with the attention network, representing a significant 7.59% improvement. This improvement can be attributed to the attention network's ability to discover crucial subsequences while learning the sequence representation, which ultimately leads to better performance.

CHAPTER 5
SAHT: STRUCTURE-AWARE HYPERGRAPH TRANSFORMER

5.1 Introduction

In this paper, we present `SaHT`, a novel Structure-aware Hypergraph Transformer model, which generates node representations using a new structure-aware self-attention mechanism that identifies the importance of nodes and hyperedges from both semantic and structural perspectives.

First, we create hypergraphs from standard graphs, preserving higher-order structural information. Our `SaHT` model then employs a learnable *structure encoding* scheme in the input layer, capturing local and global structural information through local structure encoding, centrality encoding, and uniqueness encoding. We also use a learnable hypergraph Laplacian eigenvector as a position encoder to incorporate distance-aware spatial information.

To capture the varying degrees of structural and semantic importance of nodes and hyperedges, we introduce a *structure-aware self-attention* mechanism consisting of two layers. The Local Structure-Aware Node-to-Hyperedge Attention layer aggregates node representations into hyperedge representations by emphasizing structurally and semantically significant nodes, using node-local clustering coefficient and node coreness. The Global Structure-Aware Hyperedge-to-Node Attention layer aggregates hyperedge representations into node representations by highlighting important hyperedges, using hyperedge density score and hyperedge clustering coefficient.

Our contributions are summarized as follows:

- **Hypergraph Construction:** Unlike existing methods that create hypergraphs by grouping nodes with similar attributes, potentially losing structural information, we construct hypergraphs by discerning higher-order connections between nodes, preserving node structural integrity.

- **Node Feature Enrichment via Structural and Spatial Encoding:** `SaHT` introduces three structure encoding schemes and a position encoder to enhance the capture of structural and spatial information, allowing the model to leverage both initial node features and discovered structural-spatial patterns.

- **Structure-Aware Self-Attention:** Our structure-aware self-attention mechanism incorporates both attribute-based semantic features and structural information, identifying crucial nodes for hyperedges and important hyperedges for nodes from both perspectives. We introduce four measures to discover structurally significant nodes and hyperedges.

5.2 Background

In this section, we first outline the basics of transformer. Then, we review the literature on graph and hypergraph neural networks, including their transformer variants, emphasizing key contributions and findings in these areas.

5.2.1 Preliminaries

5.2.1.1 Transformer

The transformer is a neural network architecture that exploits a self-attention mechanism to capture local and global dependencies in the input data. The main components of the model are a multi-head self-attention (MHA) module and a position-wise feed-forward network (FFN). The MHA module computes attention weights by projecting the input into a query (Q), key (K), and value (V) vectors. If $Z \in \mathbb{R}^{n \times d}$ is a d dimensional input feature where $Z = [z_1, z_2, ..., z_n]^T$, the MHA first projects it to Q, K, and V using three learnable weight matrices $W^Q \in \mathbb{R}^{d \times d_Q}$, $W^K \in \mathbb{R}^{d \times d_K}$, and $W^V \in \mathbb{R}^{d \times d_V}$ as

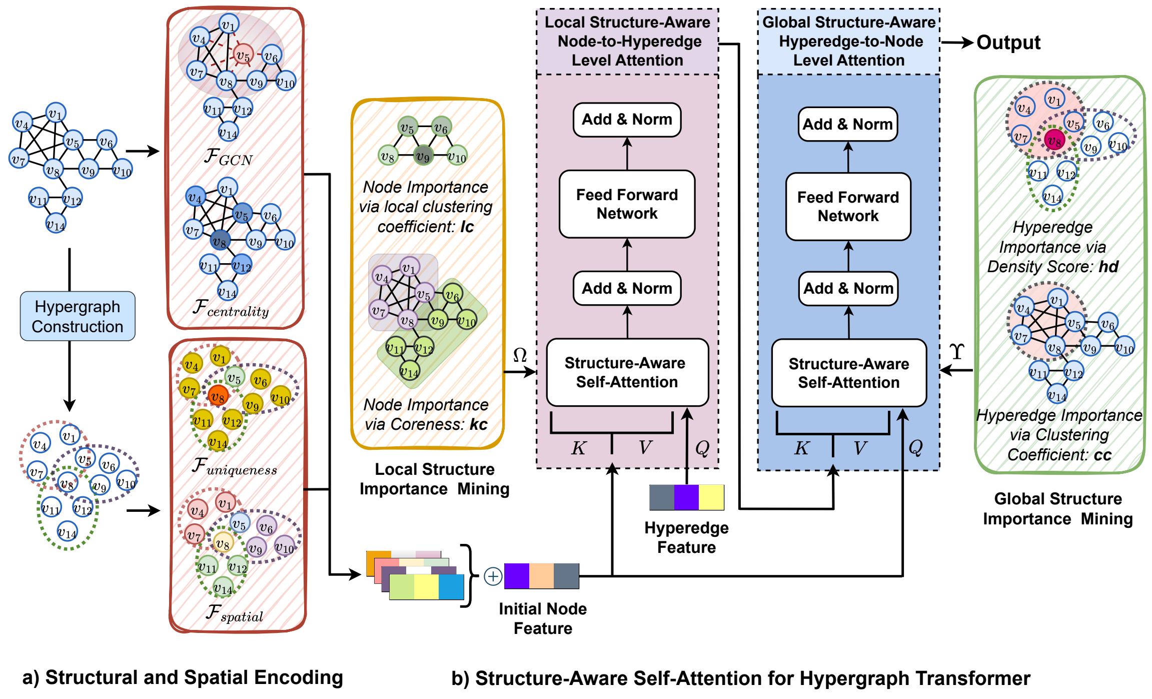
$$Q = ZW^Q, K = ZW^K, \text{ and } V = ZW^V. \tag{5.1}$$

Then in each head $h \in \{1, 2, 3, ..., H\}$, the self-attention mechanism is applied to the corresponding (Q_h, K_h, V_h) as

$$\Delta_h = (Q_h K_h^T)/\sqrt{d_K}, \tag{5.2}$$

$$output_h = \text{softmax}(\Delta_h)V_h. \tag{5.3}$$

Here, Δ represents the attention map and the dimensions of $d_Q = d_K = d_V = d$. The outputs from different heads are concatenated and transformed to get the MHA output, which is further fed into a position-wise FFN layer. Residual connections and layer normalization are used to stabilize and normalize the outputs.



Hypergraph Construction
$\mathcal{F}_{GCN}$
$\mathcal{F}_{centrality}$
$\mathcal{F}_{uniqueness}$
$\mathcal{F}_{spatial}$
Node Importance via local clustering coefficient: lc
Node Importance via Coreness: kc
Local Structure Importance Mining
Ω
Local Structure-Aware Node-to-Hyperedge Level Attention
Add & Norm
Feed Forward Network
Add & Norm
Structure-Aware Self-Attention
K
V
Q
Hyperedge Feature
Initial Node Feature
Global Structure-Aware Hyperedge-to-Node Level Attention
Output
ϒ
Hyperedge Importance via Density Score: hd
Hyperedge Importance via Clustering Coefficient: cc
Global Structure Importance Mining
a) Structural and Spatial Encoding
b) Structure-Aware Self-Attention for Hypergraph Transformer

Figure 5.1 Structure-aware Hypergraph Transformer, `SaHT`, consists of a) *Structural and Spatial Encoding* that enriches initial node representation via learnable structural and spatial node features and b) *Structure-Aware Self-Attention* that enables the integration of structural importance of nodes for a hyperedge and hyperedges for a node into regular attribute-based semantic attention to derive the ultimate node representations.

5.3 Methodology

In this section, we outline components of our `SaHT` model. As the first step of `SaHT`, we present the structure and spatial encoding module as the initial node feature enrichment process. Next, we propose a structure-aware self-attention module that integrates both the local and global structural information of nodes and hyperedges into the regular semantic feature-based self-attention network as an inductive bias.

In Section 5.3.3, we explain how we create our structure-based hypergraph from a standard graph. Unlike existing works that construct hypergraphs by grouping semantically similar nodes into hyperedges [7, 9, 8], which may result in a loss of critical structural information, we develop a hypergraph from an input graph by identifying subgraphs considering higher-order structural relations of nodes and representing these subgraphs as hyperedges. We can consider different measures to define higher-order structures such as communities, cliques, and motifs. Details about hypergraph construction are presented in Section 5.3.3. We present the system architecture of our `SaHT` in Figure 5.1.

5.3.1 Node Feature Enrichment via Structural and Spatial Encoding

In graph data, a node's significance extends beyond its individual attributes. Its position, connections, and unique characteristics within the entire graph determine its relevance. Traditional feature representation methods often miss this nuanced information, as two nodes might share attributes but differ in roles due to their connections and positions. Similarly, a uniquely attributed node might still be peripheral in the overall structure. Moreover, while we create hypergraphs from graphs, we lose the nodes' local connection information. Thus, in order to incorporate the important structural and spatial information of the nodes into their representations via a transformer, we introduce four different encoding methods from both graph and hypergraph perspectives: (1) local structure encoding lse, (2) centrality encoding ce, (3) uniqueness encoding ue and (4) position encoding pe. We combine all these encodings and integrate them into initial (i.e., 0^{th} layer) node features of a node v_i, x_i^0, as follows

$$x_i = \text{Aggregate}(x_i^0, lse_i, ce_i, ue_i, pe_i). \tag{5.4}$$

By integrating these encodings with the original input features, the transformer can better learn node representations, encoding both structural-spatial and semantic context. The subsequent sections will dig deeper into each encoding, elucidating their formulation and contributions.

5.3.1.1 Local Structure Encoding

As mentioned earlier, we create a hypergraph from a given graph by representing subgraphs as hyperedges. However, while creating a hyperedge from a subgraph, we lose the local connection information between nodes, which might be crucial for the hyperedge representation and thus the hypergraph. To preserve the local connection information, we apply GCNs to the input graph. GCNs adeptly capture and retain local connection information by propagating information across node neighborhoods, thus discerning each node's local structure encoding (lse). This encoded information is combined with the input node features.

$$lse_i = \mathcal{F}_{GCN}\left(\mathcal{N}(i); \theta\right), \tag{5.5}$$

where lse_i denotes the local structure encoding for node v_i, calculated using a GCN function, represented as $\mathcal{F}_{GCN}$. This function is applied to the node's neighborhood, $\mathcal{N}(i)$, and is parametrized by θ, a set of learnable parameters, to adaptively capture the localized information around node v_i.

5.3.1.2 Centrality Encoding

In line with our hypergraph construction strategy, we employ closeness centrality to discern the significance of nodes within the graph structure. Nodes with high closeness centrality are integral for efficient information propagation due to their shorter distances to other nodes. In `SaHT`, we leverage closeness centrality as an additional signal to enhance the input node features. To be specific, we develop a learnable centrality encoding function $\mathcal{F}_{centrality}$; given the closeness centrality scores as input; it generates a centrality embedding vector for each node. By integrating the original nodes' features with these embedding vectors, the transformer is better equipped to grasp the nodes' roles and influence within the graph. Centrality encoding ce can be formulated as

$$ce = \mathcal{F}_{\text{centrality}}(c; \psi), \tag{5.6}$$

where c is a vector of centrality scores of all the nodes, and ψ is a learnable parameter.

5.3.1.3 Uniqueness Encoding

If a node appears in multiple hyperedges, it may not possess a distinct identity within any specific hyperedge, and as a result, its significance within those hyperedges could be diminished. To encode the importance of nodes based on their appearance in different hyperedges, we introduce a uniqueness score $\sqcap$, and for node v_i, defined as

$$\sqcap_i = 1 - \frac{C_A(i)}{C_T}, \tag{5.7}$$

where $C_A(i)$ is the number of hyperedges that node v_i appears, C_T is the total number of hyperedges in the hypergraph. The more a node appears in different hyperedges, the less uniqueness the score is. We store the uniqueness scores of all the nodes into a vector u. Given the u as an input, we exploit a learnable function $\mathcal{F}_{uniqueness}$ that assigns an embedding vector to each node according to its uniqueness score, which is further added with the input node features. Uniqueness encoding ue is defined as

$$ue = \mathcal{F}_{\text{uniqueness}}\left(u; \zeta\right), \tag{5.8}$$

where u is a vector of uniqueness scores of all the nodes, and ζ is a learnable parameter.

5.3.1.4 Position Encoding

Position encoding (pe) plays a crucial role in the transformer architecture for capturing sequential information using different position functions (e.g., sine, cosine). However, when dealing with arbitrary graphs, the direct application of positional encoding becomes challenging due to the absence of a clear positional notion. To overcome this limitation, researchers have introduced the use of eigenvectors derived from the graph Laplacian as a graph-specific alternative to sine and cosine functions [109, 135].

We adopt the method in [136] to calculate the hypergraph Laplacian eigenvector ev. Then, we apply a learnable function, $\mathcal{F}_{\text{spatial}}$, which, when provided with the ev, produces an embedding for each node, serving as the node positional encoding. This captures the nodes' structural information, considers their relationships, and incorporates the connectivity patterns within the hypergraph. It also encodes distance-aware (spatial) information where nearby nodes or nodes connected by hyperedges share similar positional features. We incorporate this positional information with the node input features. Additionally, we address eigenvector multiplicity by randomly flipping the sign during training. This regularization technique prevents the model from overfitting to specific sign patterns and promotes the learning of more robust and generalized representations. Position encoding pe can be expressed as

$$pe = \mathcal{F}_{\text{spatial}}(ev; \phi), \tag{5.9}$$

where ev is hypergraph Laplacian eigenvectors and ϕ is a learnable parameter.

5.3.2 Structure-Aware Self-Attention

In this section, we describe our structure-aware self-attention mechanism capturing high-order relationships between nodes and hyperedges. This model incorporates novel two-level attention, encompassing both node-to-hyperedge and hyperedge-to-node information propagation via considering their semantic and structural importance for each other, facilitating the integration of local and global contexts into node representation learning.

The first level, node-to-hyperedge, aggregates node information to create hyperedge representations via local structure-aware self-attention. Conversely, the second level, hyperedge-to-node, gathers hyperedge representations to generate the final node representation via global structure-aware self-attention. As shown in Equation 5.2, the self-attention network in traditional transformer architecture calculates attention scores between different nodes and hyperedges by evaluating the attribute-based semantic similarity whereas the vital structural information is ignored. To address this, at each attention level, we introduce different structural inductive biases that allow the model to capture both semantic and vital structural information simultaneously.

5.3.2.1 Local Structure-Aware Node-to-Hyperedge Level Attention.

Hyperedge is degree-free and consists of an arbitrary number of nodes. However, the contribution of nodes and their importance in hyperedge construction could be different. To identify the important nodes for a given hyperedge, we leverage a self-attention mechanism that aggregates node representations and assigns higher weights to crucial ones. In addition to a semantic perspective of regular self-attention, we inherit the local structure of subgraphs as hyperedges to extract the structural significance of nodes within hyperedges. Since we represent each subgraph as a hyperedge, the structurally significant nodes for a subgraph will also be important for that corresponding hyperedge. To determine these nodes, we calculate the structural importance of nodes for that subgraph (hyperedge). Then, we combine nodes' structural importance with the attention map as a structure inductive bias. With the modified attention mechanism, the l-th layer representation q_j^l of a hyperedge e_j is defined as

$$q_j^l = \alpha \left(\sum_{v_i} [\Gamma_{ji} W_1 p_i^{l-1} \mid v_i \in e_j] \right), \tag{5.10}$$

where α is a nonlinear activation function, W_1 is a trainable weight matrix, and Γ_{ji} is the attention coefficient of node v_i in the hyperedge e_j. The attention coefficient is defined as

$$\Gamma_{ji} = \frac{\exp(r_{ji})}{\sum_{v_k}[\exp(r_{jk}) \mid v_k \in e_j]}, \tag{5.11}$$

and

$$r_{ji} = \frac{\beta(W_2 p_i^{l-1} * W_3 q_j^{l-1})}{\sqrt{d_K}} + \Omega_{ji}, \tag{5.12}$$

where β is a LeakyReLU activation function, v_k is the node that belongs to hyperedge e_j, W_2, W_3 are the trainable weight matrices, $*$ is the element-wise multiplication, Ω_{ji} is the total structural importance of node v_i for hyperedge e_j. We first start by presenting how we calculate nodes' structural importance, namely the local clustering coefficient (lc) and k-core (kc) values. We will eventually add these values to get the total local structural importance of a node for a hyperedge as

$$\Omega_{ji} = lc_{ji} + kc_{ji}. \tag{5.13}$$

i. Node Importance via Local Clustering Coefficient. Understanding the structural significance of individual nodes within a network is crucial for various applications such as information dissemination and influence assessment. One effective way to quantify a node's structural importance is by examining its local clustering coefficient. For our subgraphs, it can be defined as follows.

Definition 1 (Node local clustering coefficient)**.** *The local clustering coefficient (lc) of a node in a subgraph is the ratio of the number of connections between the node's neighboring nodes (within the subgraph) to the total number of possible connections between them (within* *the subgraph), i.e.,*

$$lc_{ji} = \frac{I_{ji}}{\frac{g_{ji}(g_{ji}-1)}{2}}.$$

In this formula, I_{ji} represents the number of connections existing among the neighbors of the node v_i within the subgraph that pertains to hyperedge e_j. Conversely, the term $\frac{g_{ji}(g_{ji}-1)}{2}$ computes the maximum possible number of connections among all neighbors of v_i, where g_{ji} indicates the degree of the node v_i in the subgraph associated with hyperedge e_j.

The lc of nodes measures the density of connections among their neighbors, indicating the presence of tightly-knit clusters. Nodes with high lc are important as they are likely to facilitate information flow, influence spreading, and contribute to network resilience.

ii. Node Importance via Coreness. The concept of k-core decomposition is rooted in the identification of a graph's maximal subgraphs, where each node is connected to at least k other nodes within the network. It assigns a k-core (kc) value to each node, representing the highest level of connectivity it shares with its neighbors. Consequently, nodes with high kc values are considered more central and pivotal within the network. The process begins by assigning a k-core value to each node of the subgraph, quantifying its connectivity level within the network. The decomposition process then systematically prunes nodes with lower connectivity, starting from those with the least connections and progressively moving towards nodes with higher degrees of connectivity. This pruning process is dynamic, with the connectivity degrees of neighboring nodes adjusted accordingly. Nodes persisting in the highest k-core constitute the network's core structure, showcasing robust connections.

5.3.2.2 Global Structure-Aware Hyperedge-to-Node Level Attention.

In hypergraphs, a node can be part of several hyperedges, yet not all hyperedges hold equal significance for a node. This necessitates a self-attention mechanism to emphasize the key hyperedges associated with a specific node. Moreover, employing vanilla self-attention as defined in Equation 5.2 will only capture the importance of hyperedges from a semantic point of view, ignoring structural importance. To address this issue, we dive into the structural properties of hyperedges, and the corresponding subgraphs and calculate the structural importance of the subgraphs (hyperedges) for each node.

After determining the structural importance of hyperedges, we combine them with the attention map of the hyperedge-to-node level attention network as a structure inductive bias. With the modified attention mechanism, the l-th layer representation p_i^l of node v_i is defined as

$$p_i^l = \alpha \left(\sum_{e_j} [\Lambda_{ij} W_4 q_j^l \mid e_j \in \mathcal{E}_i] \right), \tag{5.14}$$

where α is a nonlinear activation function, W_4 is a trainable weight matrix, $\mathcal{E}_i$ is the set of hyperedges connected to node v_i, and Λ_{ij} is the attention coefficient of hyperedge e_j on node v_i. The attention coefficient is defined as

$$\Lambda_{ij} = \frac{\exp(t_{ij})}{\sum_{e_k} [\exp(t_{ik}) \mid e_k \in E_i]}, \tag{5.15}$$

and

$$t_{ij} = \frac{\beta(W_5 q_j^l * W_6 p_i^{l-1})}{\sqrt{d_K}} + \Upsilon_{ij}, \tag{5.16}$$

Algorithm 5: Hyperedge Clustering Coefficient Computation

```
Input: Subgraph $\mathcal{H}_s = (\mathcal{V}_s, \mathcal{E}_s)$
Output: Clustering Coefficients $cc$ of $\mathcal{H}_s$
$nodes \leftarrow \text{list}(\mathcal{V}_s)$
$triangle\_count \leftarrow 0$
for $(idx, u) \in enumerate(nodes)$ do
    for $v \in nodes[idx + 1 :]$ do
        if $\mathcal{H}.has\_edge(u, v)$ then
            $com\_neig \leftarrow \text{list}(\text{common\_neighbors}(\mathcal{H}_s, u, v))$
            $triangle\_count \leftarrow triangle\_count + \text{len}(com\_neig)$
        end
    end
end
$triangle\_count \leftarrow triangle\_count/3$ ; /* Each triangle is counted three times
 */
$m_{\mathcal{H}} \leftarrow |\mathcal{V}_{\mathcal{H}}|$ $total\_possible\_triangles \leftarrow \frac{m_{\mathcal{H}}(m_{\mathcal{H}}-1)(m_{\mathcal{H}}-2)}{6}$
if $total\_possible\_triangles > 0$ then
    $clustering\_coefficient \leftarrow \frac{triangle\_count}{total\_possible\_triangles}$
end
else
    $clustering\_coefficient \leftarrow 0.0$
end
return $cc \leftarrow clustering\_coefficient$
```

where W_5 and W_6 are the trainable weight matrices, and Υ_{ij} is the total structural importance of hyperedge e_j for node v_i. To calculate the structural significance of a hyperedge for its member nodes, we introduce two new measures: hyperedge density (hd) and hyperedge clustering coefficient (cc). By computing these measures, we can determine the relative importance of hyperedges in terms of their structural impact on the connected nodes. Calculated measures are simply added to get the total global structural importance of a hyperedge for a node as

$$\Upsilon_{ij} = hd_{ij} + cc_{ij}. \tag{5.17}$$

i. Hyperedge Importance via Density Score. Understanding the significance of hyperedges within a hypergraph is essential for analyzing network connectivity and cohesiveness. One effective way to measure the importance of a hyperedge is its density score.

Definition 2 (**Hyperedge Density**)**.** *The hyperedge density (hd) quantifies the proportion of nodes that a hyperedge e_j consists of compared to the total number of nodes in the hypergraph. If a hyperedge e_j consists of m_{e_j} nodes and m is the total number of nodes in the hypergraph, then hd can be expressed as $\frac{m_{e_j}}{m}$.*

A higher *hd* indicates stronger interconnectivity among the nodes within the hyperedge, implying a more cohesive and significant grouping. Hyperedges consisting of a larger number of nodes are deemed more significant for a given node compared to hyperedges with fewer nodes.

ii. Hyperedge Importance via Clustering Coefficient. In hypergraphs, hyperedges are crucial for representing complex relationships and patterns. To evaluate the structural importance of these hyperedges, we introduce the hyperedge clustering coefficient.

Definition 3 (Hyperedge clustering coefficient)**.** *The hyperedge clustering coefficient (cc) is defined from its subgraph, which is the ratio of the number of existing triangles between all pairs of nodes to the total number of possible triangles in that subgraph.*

The *cc* provides insights into the cohesive structure of a subgraph by quantifying the extent of interconnectivity among its nodes. A higher *cc* signifies that the nodes within a subgraph are densely interconnected, indicating a strong level of cohesion. A subgraph (hyperedge) with a larger cc score is considered important for a node. Algorithm 5 outlines the steps to compute the hyperedge clustering coefficient from the subgraph.

Like the transformer paper [41], the output of each self-attention layer is fed into a position-wise FFN layer. Moreover, residual connections and layer normalization are used to stabilize and normalize the output. `SaHT` generates the node representations by utilizing these two levels of attention. Finally, the output of `SaHT` is linearly projected with a shared trainable weight matrix W_S to generate an S dimensional output for each node as $O = pW_S$, where S is the number of classes, p is the output of the `SaHT`. We train our entire model using a cross-entropy loss function.

5.3.3 Hypergraph Construction

Existing models for creating hypergraphs from standard graphs typically utilize node attributes and methods like k-means/k-NN to group semantically similar nodes into a single hyperedge [7, 9, 8]. That may cause a loss of structural insight in the graph, which is vital for effective node representation learning in a hypergraph setting. To address this challenge, We develop a hypergraph from an input graph by identifying subgraphs considering higher-order structural relations of nodes and representing these subgraphs as hyperedges. We can consider different measures to define higher-order structures such as communities, cliques, and motifs. In this study, we use communities to represent higher-order structural relations of nodes. Communities are densely connected subgraphs capturing group interaction beyond the pairwise interaction. While representing each community as a hyperedge, there should be overlaps between communities to make the hypergraph connected. To identify these commu-

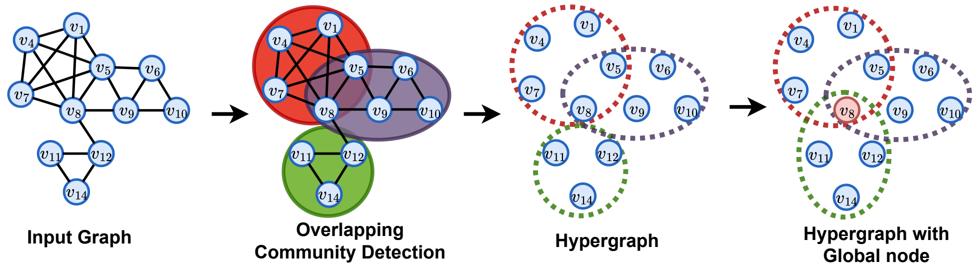


Figure 5.2 Hypergraph Construction: Each community is represented as a hyperedge.

nities, we evaluate various overlapping community detection methods and eventually adopt the algorithm described in [137], which exhibits better performance in our experiments. This algorithm utilizes edge attributes as weights; in cases where the input graph does not provide these attributes, we assign a uniform weight of 1 to each edge. Each detected community is then represented as a hyperedge in the hypergraph, with the community members serving as nodes within these hyperedges.

While employing an overlapping community detection algorithm, it's notable that not all communities will overlap, occasionally resulting in isolated hyperedges that lack connections to others. This isolation can hinder the flow of information within the hypergraph. To mitigate this and enhance connectivity between hyperedges, we strategically incorporate global nodes chosen based on their high closeness centrality scores from the input graph. Closeness centrality, measuring the average shortest distance of a node to all other nodes, helps identify nodes that can rapidly disseminate information across the network. By linking these centrally located global nodes to hyperedges, we significantly boost the interconnectivity and information exchange across the hypergraph, ensuring a more cohesive and efficient network structure. Figure 5.2 provides a detailed visual representation of the hypergraph construction process.

5.3.4 Complexity

Given the f dimensional initial feature of a node, `SaHT` exploits a two-level attention network and generates $f^{'}$ dimensional embedding vector for the node. Thus, the time complexity of `SaHT` can be expressed in terms of the cumulative complexity of the attention networks along with the structure encoding involved in each attention module.

To formulate the total time complexity for the *Local Structure-Aware Node-to-Hyperedge Level Attention*, we divide it into two parts. The first part involves calculating the time complexity of the node-to-hyperedge level attention, which can be formulated as $O(|\mathcal{V}|ff^{'} + |\mathcal{E}|d_e f^{'})$, wherein $|\mathcal{V}|$ represents the number of nodes, $|\mathcal{E}|$ represents the number of hyperedges, and d_e denotes the average size of each hyperedge, typically a minimal value. In the second part, we calculate the time complexity of the local clustering coefficient and k-core. The local clustering coefficient exhibits a time complexity of $O(|\mathcal{E}|d_e)$. Analogously, the time complexity of the k-core decomposition can be expressed as $O(|\mathcal{E}|d_e)$. So, the overall time complexity involved in *Local Structure-Aware Node-to-Hyperedge Level Attention* is $O(|\mathcal{V}|ff^{'} + |\mathcal{E}|d_e f^{'})$+$2O(|\mathcal{E}|d_e)$.

Similarly, if d_n represents the average node degree (i.e., the average number of hyperedges to which a node belongs), the time complexity for the hyperedge-to-node level attention can be expressed as $O(|\mathcal{E}|ff^{'} + |\mathcal{V}|d_n f^{'})$. The cumulative time complexity required to calculate hd is demonstrable $O(|\mathcal{V}|d_n)$. Consequently, the overarching time complexity incurred for determining the hyperedge clustering coefficient is calculated to be $O(|\mathcal{E}|(d_e c_n)^{3/2})$, where

c_n is the average number of common neighbors between any two nodes in a hyperedge. So, the overall time complexity involved in Global Structure-Aware Hyperedge-to-Node Level Attention is $O(|\mathcal{E}|ff' + |\mathcal{V}|d_n f')+O(|\mathcal{V}|d_n)+O(|\mathcal{E}|(d_e c_n)^{3/2})$.

5.4 Experiment

5.4.1 Experimental Setup

We evaluate `SaHT` on five distinct datasets including citation network and social network datasets [138]. Detailed statistics for these datasets are provided in Table 5.1. The community detection algorithm follows [137] with default settings. We also present constructed hypergraph statistics of these datasets in Table 5.1. The number of global nodes for the Cora and the Citeseer, PubMed, DBLP, and LastFMAsia (LFMA) datasets is set to 3, 1, 4, 5, and 4, respectively. Our experiments follow a standard split of 50% for training, 25% for validation, and 25% for testing.

To assess performance, we benchmark `SaHT` against sixteen models using their default settings. These models are categorized based on their architectural design: models intended for standard graphs are referred to as graph-based models, while those designed for hypergraphs are referred to as hypergraph-based models. The baseline hypergraph-based models utilize hypergraphs constructed according to the methodologies described in their respective original studies. The graph-based models include Graph Transformer (GT) full and sparse versions [109], Graphormer [42], ANS-GT [44], NAGphormer [43], GPRGNN [139], GCNII [55]. The hypergraph-based models include HGNN [7], HCHA [50], HyperGCN [79],

Dataset	**#N**	**#E**	**#C**	**#HyE***
Cora	2,708	5,429	7	263
Citeseer	3,312	4,715	6	563
PubMed	19,717	44,338	3	168
DBLP	17,716	105,734	4	739
LFMA	7,624	55,612	18	46

Table 5.1 Dataset Statistics. Here, #N, #E, and #C are the number of nodes, number of edges, and number of classes, respectively. Furthermore, #HyE* indicates the number of hyperedges in the hypergraphs derived from these datasets via our hypergraph construction method.

DHGNN [8], HNHN [140], UniGCNII [82], AllSetTransformer [83], SheafHyperGCN and SheafHyperGNN [141], HSL [142], and DHKH [143].

In the structural and spatial encoding methods described in Section 5.3.1, we employ four different encoding functions to capture distinct aspects of the input graph and constructed hypergraph. To integrate local connectivity information with lse, we utilize a two-layer GCN implemented in Deep Graph Library (DGL) [144]. To compute ce and ue, we develop learnable encoding functions by utilizing PyTorch's learnable *Embedding layer*. pe is executed using PyTorch's learnable *Linear layer* that leverages hypergraph Laplacian eigenvectors to generate positional embeddings for each node.

In `SaHT`, we conduct semi-supervised node classification in a transductive setting, repeating experiments ten times with different random splits. Node and hyperedge features are one-hot encoded, and we employ a single-layer `SaHT` with Adam optimization. The optimal hyperparameters are determined through a grid search on the validation set. Based on our grid search, we chose a learning rate of 0.001 and a dropout rate of 0.5 for regularization. The LeakyReLU activation function is applied, and the model has four attention heads. Training spans 500 epochs, incorporating early stopping if the validation accuracy does not change

Method	Model	Cora	Citeseer	PubMed	DBLP	LFMA
Graph-based	GT-full	63.10	59.05	77.02	78.85	79.23
	GT-sparse	72.04	66.12	80.11	82.15	81.21
	Graphormer	70.76	67.52	82.17	80.01	83.23
	ANS-GT	86.70	73.52	87.92	82.52	86.88
	NAGphormer	86.12	71.55	87.51	81.89	86.15
	GPRGNN	86.17	76.11	86.96	81.78	85.92
	GCNII	86.71	75.86	88.07	81.58	85.96
Hypergraph-based	HGNN	79.39	72.45	86.44	78.25	80.48
	HCHA	79.14	72.42	86.41	79.52	82.44
	HyperGCN	78.45	71.28	82.84	82.78	80.20
	DHGNN	79.52	73.59	80.50	80.37	80.22
	HNHN	76.36	72.64	86.90	80.72	84.17
	UniGCNII	78.81	73.05	88.25	82.17	84.49
	AllSetTransformer	78.59	73.08	88.72	83.15	86.21
	SheafHyperGCN	80.06	73.27	87.09	82.37	86.88
	SheafHyperGNN	81.30	74.71	87.68	83.11	87.14
	HSL	79.88	73.79	-	-	-
	DHKH	82.60	71.68	77.50	81.48	83.25
	SaHT	**88.48**	**78.09**	**89.01**	**84.38**	**88.58**

Table 5.2 Performance Comparisons: Mean accuracy (%)

for 100 consecutive epochs. The hidden dimension of SaHT is set to 64. SaHT is implemented using the DGL with PyTorch on a Tesla V100-SXM2-32GB GPU.

5.4.2 Result

We evaluate the performance of our model by conducting experiments on five distinct datasets and comparing the results with thirteen state-of-the-art baselines. The baselines are considered if their experimental results or codes are available. The outcomes, presented in Table 5.2, unambiguously demonstrate the superiority of our model across all datasets. Specifically, our model excels on the Cora dataset, achieving an impressive accuracy of 88.48%. This significantly surpasses the accuracy of the best-performing graph-based baseline model, GCNII, at 86.71% and exceeds the top-reported accuracy of the hypergraph-based baseline, DHKH, which stands at 82.60%. In the case of the Citeseer dataset, our model attains an accuracy of 78.09%, outperforming the graph-based leading baseline GPRGNN with an accuracy of 76.11%, and the hypergraph-based top-performing baseline, SheafHyperGNN, at 74.71%. The trend continues with the PubMed, DBLP, and LastFMAsia datasets, where our model substantially outperforms the baselines. The results underscore our model's substantial enhancements in classifying the datasets, setting a new standard compared to existing state-of-the-art methods.

A closer look at Table 5.2 reveals that the performance of hyper-graph-based models is promising. This success might be attributed to their ability to efficiently learn the intricate higher-order structure within the hypergraph. In general, hypergraph-based baseline models present stiff competition, with DHGNN, AllSetTransformer, and SheafHyperGNN slightly outpacing others. Specifically, DHGNN, DHKH, and HSL acknowledge that the input hypergraph structure might not adequately represent the underlying relations in the data. Consequently, they simultaneously learn the hypergraph structure and hypergraph neural network, enabling them to prune noisy task-irrelevant and false-negative connections, producing better output. On the other hand, the AllSetTransformer framework blends Deep Sets [84] and Set Transformers [85] with hypergraph neural networks to learn multiset functions. This synergy provides substantial modeling flexibility and expressive power, elevating the performance in various tasks. SheafHyperGNN and SheafHyperGCN enhance hypergraph representation by introducing cellular sheaves, a mathematical construction that adds additional structure to the conventional hypergraph while preserving their local, higher-order connectivity. This enhancement increases the expressivity of the models, enabling them to capture more nuanced and complex interactions within the hypergraph, leading to better

SaHT	Cora	Citeseer	PubMed	DBLP	LFMA
LS	83.12	73.89	85.32	81.13	83.86
GS	79.68	70.56	82.88	79.32	80.36

Table 5.3 Impact of local and global structural knowledge on the model performance (accuracy) for only local structures (LS) (first row) and only global structures (GS) (second row).

overall representation.

Hypergraphs, known for their adeptness at representing intricate higher-order relationships, have become invaluable in numerous scientific endeavors. The proposed SaHT model builds on this, representing communities as hyperedges and thus forming a hypergraph that captures intricate node relations. In contrast to existing models that do not consider local connection information during hypergraph construction, SaHT leverages GCN to preserve this vital information. Additionally, SaHT integrates unique encoding modules to learn structural-spatial information and a structure-aware self-attention module. These modules enable SaHT to produce robust node representations, recognizing key nodes and hyperedges from both structural and semantic perspectives.

5.4.2.1 Case study - Examining the impact of local and global structural knowledge

To delve deeper into the roles of local and global structural knowledge, we train SaHT excluding local structural components (specifically, lse, ce, lc, and kc) and omitting global structural elements (namely, ue, hd, and cc) on all dataset. Outcomes in Table 5.3 illustrate that local structural information significantly influences SaHT efficacy more than its global counterpart. Given that hypergraphs inherently harness higher-order global structural details from the graph, the omission of global encodings in SaHT does not critically hinder its performance. On the other hand, converting a graph to a hypergraph can result in a loss of

SaHT W/O / **Dataset**	*pe*	*ce*	*ue*	*lse*	*lc*	*kc*	*hd*	*cc*
Cora	85.51	84.19	84.93	81.50	84.64	84.34	86.21	85.12
PubMed	86.22	85.52	86.68	82.44	85.88	86.72	87.34	86.57

Table 5.4 Comparative impact of different design schemes on the model performance (accuracy) for Cora and PubMed datasets.

local structural details, underscoring the importance of integrating local structural insights into hypergraph learning.

5.4.2.2 Case study - Examining the impact of different design schemes

To better understand the impact of each design scheme, we conduct ablation studies on a small and a large dataset, Cora and PubMed, respectively. We consider running `SaHT` without (W/O) considering a specific design scheme each time. The results are presented in Table 5.4, which implies the impact of the individual design scheme on the model performance. For instance, removing positional encoding (pe) results in an accuracy of 85.51% for the Cora dataset and 86.22% for the PubMed dataset. This table shows that for both datasets, the greatest performance degradation occurs when the local structure encoding (lse) is excluded from the input node feature. When we create hyperedge from a community, we lose the local connection information between the nodes, which might be very important for the hyperedges. lse preserves local connection information by applying GCN to the input graph. The significant decrease in performance without lse highlights its crucial contribution to the model's overall performance.

Model	Cora	Citeseer	PubMed	DBLP	LFMA
SaHT	88.48	**78.09**	**89.01**	84.38	**88.58**
$\texttt{SaHT}_{\text{N}}$	86.88	76.23	88.41	**85.46**	87.26
$\texttt{SaHT}_{\text{C}}$	**89.13**	77.83	87.18	79.32	87.78

Table 5.5 Impact of feature normalization and concatenation on the model performance (accuracy).

5.4.2.3 Case Study - Examining the Impact of Feature Normalization and Concatenation

As illustrated in Equation 4, our method combines various structural and spatial features (i.e., encodings) with the initial features of the nodes by summing them. To investigate the effect of normalizing these features before combining them with the node's initial features, we conduct an additional experiment. In this experiment, we normalize the features prior to their integration. The results of this experiment are presented in Table 5.5 and are labeled as $\texttt{SaHT}_{\text{N}}$. Additionally, we perform another experiment to evaluate the impact of concatenating the features instead of summing them. The outcomes of this approach are also shown in Table 5.5 and are referred to as $\texttt{SaHT}_{\text{C}}$. A comparative analysis of the different versions of SaHT presented in the table reveals that each version performs better on different datasets. However, the original SaHT method generally outperforms both the normalized variant ($\texttt{SaHT}_{\text{N}}$) and the concatenated variant ($\texttt{SaHT}_{\text{C}}$).

5.4.2.4 Case study - Examining the impact of global nodes

We investigate the effect of global nodes (n_g) on the model's performance, illustrated in Figure 5.3. As per Figure 5.3 (a), the Cora dataset reveals a rising trend in accuracy as the global nodes n_g increase, peaking at 88.48% with $n_g = 3$, followed by a decline. Analogously, Figure 5.3 (b) presents the Citeseer dataset, where an increment in global nodes prompts an

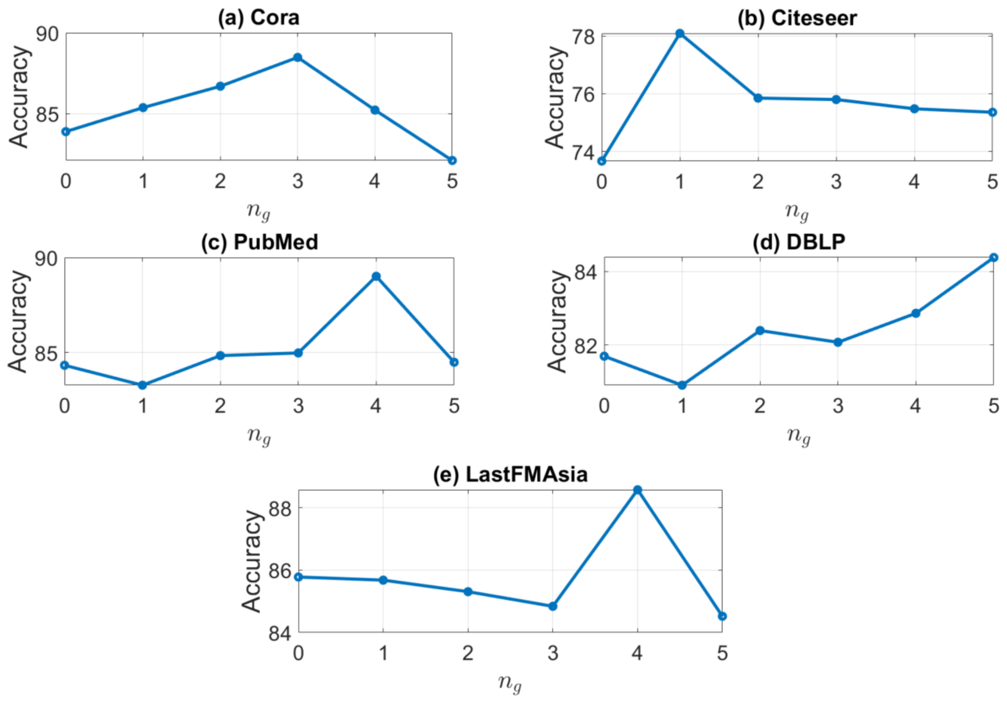


Figure 5.3 The performance of `SaHT` with different numbers of global nodes (n_g).

upsurge in accuracy, achieving a maximum of 78.09% at $n_g = 1$ before the accuracy trend reverses. Similarly, we get the best performance for the PubMed, DBLP, and LastFMAsia datasets for $n_g = 4$, $n_g = 5$, and $n_g = 4$, respectively. Overall, this trend suggests that an optimal number of global nodes can effectively incorporate relevant global information. However, exceeding this optimal number may introduce excessive parameters and reduced generalization ability. Thus, selecting an appropriate number of global nodes is crucial for optimal performance.

CHAPTER 6
HYPERGCL: GRAPH CONTRASTIVE LEARNING VIA LEARNABLE AUGMENTED HYPERGRAPH VIEWS

6.1 Introduction

In this paper, we model `HyperGCL`, an Attribute-Structure aware Graph Contrastive Learning framework from a Hypergraph perspective. To capture various granularities of higher-order information from graphs, we design three hypergraphs based on the input graph and its attributes: an attribute-driven hypergraph, a local structure-infused hypergraph, and a global structure-infused hypergraph. The attribute-driven hypergraph groups semantically similar nodes, capturing semantic similarities but potentially losing structural details. The structure-infused hypergraphs preserve local and global structural information.

Instead of predefined augmentations, we use an adaptive technique with a learnable Gumbel-Softmax function, introducing controlled stochasticity and enhancing training diversity. This dynamic augmentation improves the quality and discriminative power of the views by selectively highlighting important relationships.

We apply view-specific encoders to the augmented views. For the attribute-driven hypergraph, we use the Hypergraph Attention Network (`HyGAN`) to learn node embeddings through a two-layer attention network. To extract critical structural information from the structure-infused hypergraphs, we design Structure-aware `HyGAN` (`SHyGAN`), which incorporates node structure encodings and structural inductive biases in the attention networks.

Unlike traditional GCL methods using computer vision contrastive losses like InfoNCE or NT-Xent, we introduce a novel network-aware contrastive loss, `NetCL`. This loss extends NT-Xent by using network topology as supervised signals to define multiple positive pairs per anchor. These multiple positives are drawn from the same node in different hypergraph views, the neighbors of the anchor within a hypergraph view, or from a different hypergraph view, or the neighbors of the anchor within the input graph. To efficiently manage negative samples, we propose distance-based and similarity-based negative sampling strategies.

The contributions of this work are:

- **View Generation:** `HyperGCL` constructs three different hypergraph views capturing various granularities of information from the input graph and its attributes, addressing limitations of existing GCL approaches that mainly focus on local structure.

- **Adaptive View Augmentation:** `HyperGCL` employs a learnable Gumbel-Softmax function to adaptively augment each hypergraph view, ensuring robust training samples and overcoming the limitations of predefined augmentations.

- **View-Specific Encoder:** `HyperGCL` uses view-specific encoders, leveraging `HyGAN` for attribute-driven hypergraphs and introducing `SHyGAN` for structure-infused hypergraphs to capture critical structural information effectively.

- **Network-Aware Contrastive Loss (`NetCL`):** We propose `NetCL`, which uses network structure to define positive and negative samples, and introduce strategies to reduce the number of negative samples, lowering memory and computational costs.

6.2 Preliminaries

Hypergraphs provide a more versatile framework for representing relationships among entities than traditional graphs. Unlike graphs, which only allow edges between two nodes, hypergraphs feature hyperedges that can connect multiple nodes, accommodating varying degrees of relationships. State-of-the-art approaches to encoding hypergraph structures include different variants of Hypergrpah Neural Network (HyperGNN) such as HGNN [7], HAN [9], etc. These methods map the hypergraph to a D-dimensional latent space via a function $f : \mathcal{A} \rightarrow \mathbb{R}^D$. This mapping is achieved through higher-order message passing, effectively capturing the intricate relationships within the hypergraph. Motivated by advancements in learning from images, NLP, and graphs, we adopt CL with HyperGNN to further improve node embeddings. The main components of our `HyperGCL`, include (i) hypergraph view generation and augmentations for creating contrasting samples, (ii) view-specific hypergraph encoders, and (iii) a network-guided contrastive loss that preserves network structure for optimization. The overall pipeline is shown in Figure 6.1.

6.3 Methodology

In this section, we outline the components of our proposed `HyperGCL` model. First, we describe the three different hypergraph view generation processes from the input graph. Next, we explain the Gumbel-Softmax-based adaptive view augmentation process, followed by an overview of the hypergraph encoders. Finally, we introduce our proposed network-aware contrastive loss.

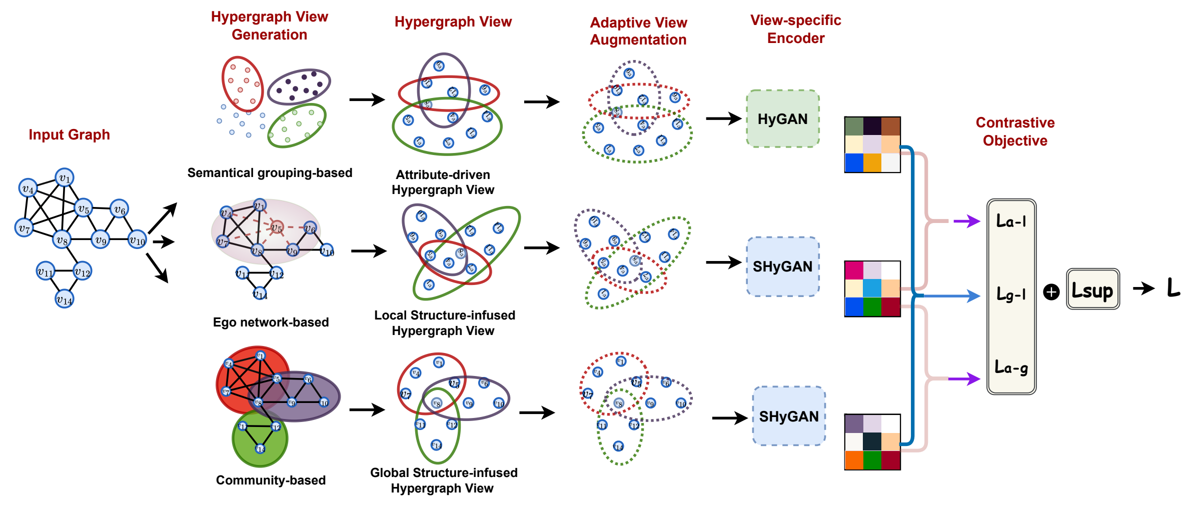

Input Graph
Hypergraph View Generation
Semantical grouping-based
Ego network-based
Community-based
Hypergraph View
Attribute-driven Hypergraph View
Local Structure-infused Hypergraph View
Global Structure-infused Hypergraph View
Adaptive View Augmentation
View-specific Encoder
HyGAN
SHyGAN
SHyGAN
Contrastive Objective
La-l
Lg-l
La-g
Lsup
L

Figure 6.1 System architecture of `HyperGCL`. The first step is constructing three different hypergraph views from the input graph and its node attributes. Then it exploits a learnable view augmentation technique to generate adaptive views. View-specific encoders are used to learn each view and finally, a network-aware contrastive loss is used with a supervised loss to train the model in an end-to-end fashion.

6.3.1 Hypergraph View Generation

Given an input graph $G = (V, E)$ where V is the set of nodes and E is the set of edges, with $m = |V|$. We begin by creating multiple hypergraph views from the G and its node attributes $\mathbf{X}$, capturing various granularities of information. Specifically, we create attribute-driven, local structure-infused, and global structure-infused hypergraph views to preserve both attribute and structural information comprehensively.

6.3.1.1 Attribute-driven Hypergraph View ($\mathcal{H}^a$)

Graph data is typically used to model pairwise relationships between nodes. However, graphs may also contain hidden higher-order complex relations that are not adequately captured by simple pairwise interactions. To capture these complex higher-order relations, we construct a hypergraph exploiting node attributes, named *attribute-driven hypergraph view*.

In this process, we group semantically similar nodes into hyperedges. To accomplish this, we apply the k-nearest neighbors (k-NN) and k-means clustering algorithms on the node attributes $\mathbf{X}$. Each node $v_i \in V$ and its k nearest neighbors form the initial hyperedges. Formally, for each node v_i, a hyperedge $\bar{e_j}$ is formed, where

$$\bar{e_j} = \{v_i\} \cup \{v_k \in V \mid v_k \text{ is } k \text{ nearest neighbors of } v_i\}.$$

Additionally, all the clusters obtained from k-means are used as additional hyperedges. Each cluster C_c from k-means clustering is considered a hyperedge:

$$\overline{\overline{e_c}} = C_c$$

Thus, each node v_i belongs to both its k nearest neighbors hyperedge and its s closest clusters hyperedges, where s closest clusters are selected based on the smallest Euclidean distances between the node and the cluster centers. Formally, let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be the set of clusters obtained from k-means. For each node v_i, the set of hyperedges it belongs to is given by:

$$e_j^a = \{\bar{e_j} \mid v_i \in \bar{e_j}\} \cup \{\overline{\overline{e_j}} \mid v_i \in C_j \text{ and}$$
$$C_j \text{ is one of the } s \text{ closest clusters to } v_i\}.$$

By combining these hyperedges, we form the hypergraph $\mathcal{H}^a = (V, \mathcal{E}^a)$, where $\mathcal{E}^a$ is the set of all hyperedges. This attribute-driven hypergraph effectively captures the higher-order relationships between nodes based on their semantic similarities, allowing for more detailed analysis and modeling of the data's complex interactions.

While this approach is effective in capturing semantic similarities of nodes, it may result in a loss of detailed structural information in the standard graph. To capture the nodes' local and global structure, we construct *local structure-infused hypergraph view* and *global structure-infused hypergraph view*.

6.3.1.2 Local Structure-infused Hypergraph View ($\mathcal{H}^l$)

The goal of constructing this hypergraph view is to capture the local structure of each node in the input graph. To achieve this, we extract a subgraph for each node, specifically considering the node and its ego network (1-hop neighbors). This subgraph is then represented by a hyperedge. In the input graph G, each node $v_i \in V$ is connected to its neighbors through edges in E. For each node $v_i \in V$, we identify its immediate neighbors and create a hyperedge e_j^l that includes v_i and all its immediate neighbors. Formally, the construction of a hyperedge for a node v_i is defined as follows:

$$e_j^l = \{v_i\} \cup \{v_k \in V \mid (v_i, v_k) \in E\}.$$

Then we combine all the hyperedges to form the hypergraph $\mathcal{H}^l = (V, \mathcal{E}^l)$, where $\mathcal{E}^l$ is the set of all hyperedges generated from the nodes' ego-networks. This approach effectively captures the local context of each node within the hypergraph structure, allowing for more detailed analysis and modeling of the graph's local connectivity patterns.

6.3.1.3 Global Structure-infused Hypergraph View ($\mathcal{H}^g$)

Unlike the hyperedges in $\mathcal{H}^l$, which are constructed based on the local structure of nodes limited to their ego-network (1-hop neighbors), the hyperedges in $\mathcal{H}^g$ are designed to capture a more comprehensive global perspective from the input graph G. This involves identifying subgraphs that account for higher-order structural relationships among nodes and representing these subgraphs as hyperedges. Various measures can be used to define these higher-order structures, such as communities, cliques, and other configurations. In this study, we utilize communities to represent the higher-order structural relationships among nodes. Communities are subgraphs with dense connections that encapsulate group interactions beyond simple pairwise interactions.

To construct the global structure-infused hypergraph, we identify communities within the graph G. Formally, let $\mathcal{CM} = \{CM_1, CM_2, \ldots, CM_k\}$ be the set of identified communities. Each community CM_c is then represented as a hyperedge, with the community members acting as nodes within these hyperedges:

$$\overline{\overline{\overline{e}}}_c = CM_c$$

When representing each community as a hyperedge, it is crucial to ensure overlaps between communities to maintain the connectivity of the hypergraph. To identify these communities, we explore various overlapping community detection methods and ultimately adopt the algorithm described in [137], as it demonstrated superior performance in our experiments. This algorithm uses edge attributes as weights. In cases where the input graph lacks these attributes, we assign a uniform weight of 1 to each edge. By combining all the hyperedges, we form the hypergraph $\mathcal{H}^g = (V, \mathcal{E}^g)$, where $\mathcal{E}^g$ represents the set of all hyperedges.

It is important to note that even after applying the overlapping community detection algorithm, some communities may remain isolated, resulting in hyperedges that are not interconnected. This isolation can impede the flow of information within the hypergraph. To address this and enhance connectivity, we incorporate global nodes selected based on their high closeness centrality scores from the input graph. Closeness centrality, which measures the average shortest distance of a node to all other nodes, helps identify nodes that can efficiently disseminate information across the network. By linking these centrally located global nodes to hyperedges, we significantly improve the interconnectivity and information exchange across the hypergraph, ensuring a more cohesive and efficient network structure.

6.3.2 Adaptive View Augmentation

For each view, we employ a learnable Gumbel-Softmax function to perform adaptive augmentation. Initially, logits representing the probability of node associations with hyperedges are initialized as learnable parameters. To introduce controlled stochasticity, these logits are perturbed with Gumbel noise and subsequently processed through a Softmax function, yielding a probabilistic mask. A hard threshold is then applied to the probabilistic mask to derive a binary mask, $\mathbf{m}$, indicating the inclusion of nodes in the hyperedges for the augmented view. To ensure differentiability, we use the straight-through estimator technique. This technique allows gradients to propagate through the Softmax probabilities by combining the binary mask $\mathbf{m}$ with the probabilistic mask $\mathbf{p}$. Formally, for each node v_i, the logits

ϕ_{v_i} are perturbed with Gumbel noise ϵ, and the Softmax function is applied as follows:

$$\mathbf{p}_{v_i} = \text{softmax}\left(\frac{\phi_{v_i} + \epsilon}{\tau}\right), \tag{6.1}$$

where τ is the temperature parameter. The binary mask for node v_i, $\mathbf{m}_{v_i}$, is obtained by applying a threshold θ to $\mathbf{p}_{v_i}$:

$$\mathbf{m}_{v_i} = (\mathbf{p}_{v_i} > \theta). \tag{6.2}$$

To ensure gradients propagate through the Softmax probabilities $\mathbf{p}_{v_i}$ instead of the binary mask $\mathbf{m}_{v_i}$, we employ the straight-through estimator as follows:

$$\tilde{\mathbf{m}}_{v_i} = (\mathbf{m}_{v_i} - \mathbf{p}_{v_i}) + \mathbf{p}_{v_i}. \tag{6.3}$$

The final augmented hypergraph view is produced by element-wise multiplying the binary mask matrix $\mathbf{M}$, composed of all $\tilde{\mathbf{m}}$, with the original hypergraph incidence matrix $\mathbf{A}$:

$$\tilde{\mathbf{A}} = \mathbf{M} \odot \mathbf{A}. \tag{6.4}$$

This adaptive augmentation technique also works as a method for refining the hypergraph views. By leveraging this learnable Gumbel-Softmax-based augmentation strategy, our approach ensures the generation of diverse samples, enhancing the effectiveness of CL in hypergraph settings.

After calculating the gradients during the backward pass, we update the logits of these matrices using the captured gradients. Additionally, we use the gradient information to guide the initialization of the mask matrices at each iteration, ensuring that the model starts from a more informed state. This iterative update process is expressed as:

$$\mathbf{\Theta}^{(t+1)} = \mathbf{\Theta}^{(t)} - \lambda \cdot \nabla_{\mathbf{\Theta}} \mathcal{L}^{(t)}, \tag{6.5}$$

where $\mathbf{\Theta} = \{\phi_{v_i} \mid v_i \in V\}$ represents the set of all logits, t refers to the current iteration of the training process, λ is the learning rate, and $\nabla_{\mathbf{\Theta}} \mathcal{L}^{(t)}$ denotes the gradient of the loss function $\mathcal{L}$ at iteration t with respect to the logits. By iteratively refining the logits through this gradient-based update and using gradient information to guide the initialization, we ensure continuous improvement in the quality of the probabilistic masks and, consequently, the augmented hypergraph views.

6.3.3 View-Specific Hypergraph Encoder

To generate node embeddings from each view, we utilize view-specific encoders that capture different granularities of information. Motivated by [10, 73], initially, we design and employ the Hypergraph Attention Network (`HyGAN`) on the attribute-driven hypergraph view focusing on capturing attribute-based semantic information. `HyGAN` accomplish this by employing a two-level attention mechanism: *node-to-hyperedge level attention* and *hyperedge-to-node level attention*.

`HyGAN`: Hyperedge consists of multiple number of nodes. However, all the nodes might not be equally important for a hyperedge. In `HyGAN`, *Node-to-hyperedge level attention* aggregates node information to produce hyperedge representations. It employs an attention mechanism to discover the semantically important nodes for that hyperedge and assign greater weight to them in the aggregation process. The representation of a hyperedge e_j at the l-th layer, denoted as $q^l_{e_j}$, is defined as follows:

$$q^l_{e_j} = \alpha \left(\sum_{v_i \in e_j} \left[\frac{\exp(r_{ji})}{\sum_{v_k \in e_j} \exp(r_{jk})} \right] \mathbf{W}_1 p^{l-1}_{v_i} \right), \tag{6.6}$$

where α is a nonlinear activation function, all the $\mathbf{W}$s are trainable weight matrices, and r_{ji} is calculated as

$$r_{ji} = \beta(\mathbf{W}_2 p^{l-1}_{v_i} \odot \mathbf{W}_3 q^{l-1}_{e_j}), \tag{6.7}$$

with β being a LeakyReLU activation function, $\odot$ representing hadamard product.

Similarly, a node may belong to multiple hyperedges but not all hyperedges are equally important for that node. In `HyGAN`, *Hyperedge-to-node level attention* aggregates hyperedges to generate node representation. It employs an attention mechanism to discover semantically important hyperedges for that node and assign greater weight to them in the aggregation process. The l-th layer representation $p^l_{v_i}$ of node v_i is defined as

$$p^l_{v_i} = \alpha \left(\sum_{e_j \in \mathcal{E}_{v_i}} \left[\frac{\exp(y_{ij})}{\sum_{e_k \in \mathcal{E}_{v_i}} \exp(y_{ik})} \right] \mathbf{W}_4 q^l_{v_j} \right), \tag{6.8}$$

where $\mathcal{E}_{v_i}$ is the set of hyperedges connected to node v_i, and y_{ij} is calculated as

$$y_{ij} = \beta(\mathbf{W}_5 q^l_{e_j} \odot \mathbf{W}_6 p^{l-1}_{v_i}). \tag{6.9}$$

`SHyGAN`: While `HyGAN` focuses on attribute-based semantic features to identify important nodes and hyperedges, this approach can lead to a loss of structural information when applied directly to the structure-infused hypergraph. To address this limitation, we introduce a specialized variant of `HyGAN` called Structure-aware `HyGAN` (`SHyGAN`). `SHyGAN` incorporates learnable node structure encodings to enhance the initial node features. Additionally, `SHyGAN` extends `HyGAN` by introducing a two-level topology-guided attention network. This network leverages structural inductive biases in the attention layers to identify significant nodes and hyperedges from both semantic and structural perspectives.

Node Structure Encoding: A node's significance in graph data is defined by its connectivity and role within the graph's structure, not just its individual features. Standard techniques often miss these distinctions, as nodes with similar attributes can differ greatly due to their network connections. To capture these structural details, we introduce three novel embedding techniques: (1) Local Connectivity Encoding (lce), (2) Centrality Encoding (ce), and (3) Distinctiveness Encoding (de). These are combined with the initial node features $x^0_{v_i}$ for node v_i as follows:

$$x_{v_i} = \text{Aggregate}(x^0_{v_i}, lce_{v_i}, ce_{v_i}, de_{v_i}), \tag{6.10}$$

Integrating these encodings with original features allows `SHyGAN` to learn node representations that reflect both structural and semantic contexts. Detailed explanations of each encoding method are given below.

6.3.3.1 Local Connectivity Encoding

Converting subgraphs into hyperedges loses crucial local connectivity among nodes, which might be essential for accurate hypergraph representation. To retain this, we use GCNs, that effectively capture local connectivity by aggregating information from each node's neighbors. This results in a local connectivity encoding (lce), which is then combined with the original node features to enrich the overall representation.

$$lce_{v_i} = \mathcal{G}\text{conn}\left(v_i, \mathbb{N}(v_i); \Phi\right), \tag{6.11}$$

where lce_{v_i} denotes the local connectivity encoding for node v_i, derived using the function $\mathcal{G}_{\text{conn}}$, which processes the neighborhood $\mathbb{N}(v_i)$, and Φ represents the set of trainable parameters

6.3.3.2 Centrality Encoding

Closeness centrality is a measure that captures how close a node is to all other nodes in a graph. Nodes with high closeness centrality scores can quickly interact with all other nodes, facilitating efficient information dissemination across the graph. In `SHyGAN`, we enhance node features using a learnable centrality encoding function, $\mathcal{G}_{\text{central}}$, which generates embedding vectors from centrality scores. By integrating these embeddings with the original node features, the model better understands nodes' roles and influence. Centrality encoding ce is defined as:

$$ce = \mathcal{G}_{\text{central}}(c; \psi), \tag{6.12}$$

where c is the vector of centrality scores, and ψ is a learnable parameter.

6.3.3.3 Distinctiveness Encoding

Nodes appearing in multiple hyperedges may lose distinctiveness, reducing their significance. We define a Distinctiveness score d for node v_i as:

$$d_{v_i} = 1 - (|\mathcal{E}_{v_i}|/|\mathcal{E}|), \tag{6.13}$$

where $|\mathcal{E}_{v_i}|$ is the number of hyperedges node v_i belongs to, and $|\mathcal{E}|$ is the total number of hyperedges. Higher counts result in lower Distinctiveness scores. The Distinctiveness scores are stored in a vector d. Using d, a learnable function $\mathcal{G}_{\text{Distinct}}$ generates an embedding for each node, combined with the node's initial features. Distinctiveness encoding de is given by:

$$de = \mathcal{G}_{\text{Distinct}}(d; \zeta), \tag{6.14}$$

where d is the vector of Distinctiveness scores, and ζ is a learnable parameter.

Topology-Guided Attention Network: `HyGAN` identifies semantically important nodes and hyperedges using attribute information but misses structurally important ones. To address this, we design a topology-guided attention network that employs structural inductive biases in the attention layers, enabling the model to identify key nodes and hyperedges from both semantic and structural perspectives. We define two structural inductive biases: Node Local Clustering Coefficient and Hyperedge Density to identify important nodes within a hyperedge and significant hyperedges for a node, respectively.

i. Node Significance via Local Clustering Coefficient. In network analysis, determining the importance of individual nodes is key for applications such as information dissemination and influence measurement. A useful metric for this purpose is the local clustering coefficient, which assesses the degree to which a node's neighbors are interconnected. For our subgraphs, it can be defined as follows.

Definition 4 (Node Local Clustering Coefficient)**.** *For a given node within a subgraph, the local clustering coefficient (lc) is defined as the proportion of the actual connections among* *its neighbors to the maximum possible connections that could exist among those neighbors within the subgraph. Mathematically, it is represented as*

$$lc_{ji} = \frac{2I_{ji}}{g_{ji}(g_{ji} - 1)},$$

In this formula, I_{ji} denotes the actual number of links among the neighbors of node v_i within the subgraph tied to hyperedge e_j. The expression $\frac{g_{ji}(g_{ji}-1)}{2}$ calculates the maximum potential number of links that could exist among all neighbors of v_i, where g_{ji} is the degree of node v_i within the subgraph linked to hyperedge e_j.

The local clustering coefficient lc measures how densely connected a node's neighbors are, revealing the presence of closely-knit clusters. Nodes with a high lc are seen as pivotal since they likely enhance information dissemination, influence propagation, and the robustness of the network. We incorporate lc as a structural inductive bias into Equation 6.7 as follows:

$$r_{ji} = \beta(\mathbf{W}_2 p_{v_i}^{l-1} \odot \mathbf{W}_3 q_{e_j}^{l-1}) + lc_{ji}. \tag{6.15}$$

i. Hyperedge Significance via Density Score Understanding the structural importance of hyperedges is essential for analyzing the connectivity and cohesion within a hypergraph. One useful metric for determining the significance of a hyperedge is its density score.

Definition 5 (**Hyperedge Density**)**.** *The hyperedge density (hd) measures the fraction of nodes within a hyperedge e_j relative to the total number of nodes in the hypergraph. If a hyperedge e_j contains m_{e_j} nodes and the hypergraph has a total of m nodes, the hd is given by $hd = \frac{m_{e_j}}{m}$.*

A higher hd value signifies greater interconnectivity among the nodes within the hyperedge, indicating a more cohesive and significant group. Hyperedges with more nodes are considered more influential for a specific node than those with fewer nodes. We integrate hd as a structural inductive bias into Equation 6.9 as follows:

$$y_{ij} = \beta(\mathbf{W}_5 q^l_{e_j} \odot \mathbf{W}_6 p^{l-1}_{v_i}) + hd_{ij}. \tag{6.16}$$

We apply `HyGAN` to the attribute-driven hypergraph view to generate the node embedding matrix $\mathbf{Z}^a$. Similarly, we apply `SHyGAN` to a local structure-infused hypergraph view and a global structure-infused hypergraph view, generating the node embedding matrices $\mathbf{Z}^l$ and $\mathbf{Z}^g$, respectively. These embeddings are then utilized in contrastive loss functions to preserve different granularities of information.

6.3.4 Network-Aware Contrastive Loss (`NetCL`)

Unlike previous GCL methods that directly utilize contrastive losses originally proposed in computer vision (e.g., InfoNCE [74] or NT-Xent [55]), we devise a new network-aware contrastive loss, termed `NetCL`. `NetCL` is a novel extension of the NT-Xent loss, incorporating network topology as supervised signals to define positive and negative samples in `HyperGCL`. Specifically, instead of forming only a single positive pair per anchor as in NT-Xent, `NetCL` allows for multiple positives per anchor. These multiple positives are defined as follows:

Positive Samples (`PosS`) for a node v_i include the same node v_i in two different views, nodes that are neighbors of v_i within the input graph, and nodes that belong to the same hyperedges as v_i in at least one of the views. Conversely, **Negative Samples** (`NegS`) for a node v_i include all other nodes that do not meet these criteria. Mathematically, they can be defined as:

$$
\begin{aligned}
\texttt{PosS}_{v_i} =& \{\text{same node in two different views}\} \\
& \cup \{v_j \mid v_j \text{ is a neighbor of } v_i \text{ in the input graph}\} \\
& \cup \{v_j \mid v_j \text{ belongs to the same hyperedges as } v_i \\
& \quad \text{in one of the views}\} \\
\texttt{NegS}_{v_i} =& \{\text{otherwise}\}
\end{aligned}
$$

Considering all these `NegS` instances is computationally expensive. To address this, we propose two selective negative sampling strategies: *distance-based negative sampling* and *similarity-based negative sampling.*

Definition 6 (Distance-Based Negative Sampling)**.** *For an anchor node v_i, we select the top 'a' nodes from $\texttt{NegS}_{v_i}$ that are the most distant from the anchor in the input graph. Let $dis(v_i, v_k)$ represent the distance from node v_i to node v_k. The set of distance-based negative samples $\mathcal{N}_{dis}(v_i)$ for anchor node v_i can be expressed as:*

$$\mathcal{N}_{dis}(v_i) = \{v_{k_1}, \ldots, v_{k_a} \mid dis(v_i, v_{k_1}) \geq \cdots \geq dis(v_i, v_{k_a})\},$$

where $v_{k_1}, \ldots, v_{k_a} \in \texttt{NegS}_{v_i}$ are the top 'a' most distant nodes from v_i.

Definition 7 (**Similarity-Based Negative Sampling**)**.** *In this strategy, we select the top 'a' nodes from $\texttt{NegS}_{v_i}$ that are the least similar to anchor node v_i in terms of cosine similarity. Let $sim(v_i, v_k)$ represent the cosine similarity. The set of similarity-based negative samples $\mathcal{N}_{sim}(v_i)$ for the anchor node v_i can be expressed as:*

$$\mathcal{N}_{sim}(v_i) = \{v_{k_1}, \ldots, v_{k_a} \mid sim(v_i, v_{k_1}) \leq \cdots \leq sim(v_i, v_{k_a})\},$$

where $v_{k_1}, \ldots, v_{k_a} \in \texttt{NegS}_{v_i}$ are the top 'a' least similar nodes to v_i. Using these definitions, the contrastive loss can be applied with either distance-based or similarity-based negative samples, providing a comprehensive approach to estimating the mutual information between node embeddings.

In this research, we employ three distinct contrastive learning modules to capture and preserve various granularities of information within the node embeddings produced by the encoders. These modules are as follows: i) Contrast between the attribute-driven view and the local structure-infused view, ii) Contrast between the global structure-infused view and the attribute-driven view, iii) Contrast between the local structure-infused view and the global structure-infused view.

After obtaining the node embeddings $\mathbf{Z}^a$ and $\mathbf{Z}^l$ from attribute-driven and local structure-infused hypergraphs, respectively, we adopt InfoNCE [145] to estimate the lower bound of the mutual information between them. By defining positive and negative samples, the contrastive loss function can be expressed as follows:

$$\mathcal{L}_{\text{a-l}} = -\frac{1}{m} \sum_{v_i \in V} \log \left(\frac{\sum_{v_j \in \texttt{PosS}_{v_i}} e^{\text{sim}(\mathbf{z}^a_{v_i}, \mathbf{z}^l_{v_j})/\eta}}{\sum_{v_j \in (\texttt{PosS}_{v_i} \cup \texttt{NegS}_{v_i})} e^{\text{sim}(\mathbf{z}^a_{v_i}, \mathbf{z}^l_{v_j})/\eta}} \right). \tag{6.17}$$

Where η is a temperature parameter. Similarly, the contrastive loss for contrasting the node representation from the global structure-infused view $\mathbf{Z}^g$ with the local structure-infused view $\mathbf{Z}^l$ can be expressed as:

$$\mathcal{L}_{\text{g-l}} = -\frac{1}{m} \sum_{v_i \in V} \log \left(\frac{\sum_{v_j \in \texttt{PosS}_{v_j}} e^{\text{sim}(\mathbf{z}^g_{v_i}, \mathbf{z}^l_{v_j})/\eta}}{\sum_{v_j \in (\texttt{PosS}_{v_i} \cup \texttt{NegS}_{v_i})} e^{\text{sim}(\mathbf{z}^g_{v_i}, \mathbf{z}^l_{v_j})/\eta}} \right). \tag{6.18}$$

Finally, we employ contrastive learning between the node representation from the attribute-driven view $\mathbf{Z}^a$ and the global structure-infused view $\mathbf{Z}^g$, as defined below:

$$\mathcal{L}_{\text{a-g}} = -\frac{1}{m}\sum_{v_i \in V} \log\left(\frac{\sum_{v_j \in \texttt{PosS}_{v_i}} e^{\text{sim}(\mathbf{z}^a_{v_i}, \mathbf{z}^g_{v_j})/\eta}}{\sum_{v_j \in (\texttt{PosS}_{v_i} \cup \texttt{NegS}_{v_i})} e^{\text{sim}(\mathbf{z}^a_{v_i}, \mathbf{z}^g_{v_j})/\eta}}\right). \tag{6.19}$$

Thus, the total contrastive loss L_{con} can be expressed as:

$$\mathcal{L}_{\text{con}} = \mathcal{L}_{\text{a-l}} + \mathcal{L}_{\text{g-l}} + \mathcal{L}_{\text{a-g}}. \tag{6.20}$$

Here, as $\texttt{NegS}_{v_i}$ we use $\mathcal{N}_{dis}(v_i)$ or $\mathcal{N}_{sim}(v_i)$. Their impact on the overall model performance is described in Table 6.2. We combine this contrastive loss with our supervised loss L_{sup}, which is a simple cross-entropy loss. Thus the total loss L can be expressed as:

$$\mathcal{L} = \mathcal{L}_{\text{con}} + \mathcal{L}_{\text{sup}}. \tag{6.21}$$

Dataset	**#N**	**#E**	**#C**	**#$\mathcal{E}^a$**	**#$\mathcal{E}^l$**	**#$\mathcal{E}^g$**
Cora	2,708	5,429	7	2758	2708	263
Citeseer	3,312	4,715	6	3362	3312	563
Wiki	2,405	17,981	17	2455	2405	59
PT	1,912	64,510	2	1962	1912	112
LFMA	7,624	55,612	18	7674	7624	46

Table 6.1 Dataset Statistics. #N, #E, and #C represent the number of nodes, edges, and classes, respectively. Additionally, #$\mathcal{E}^a$, #$\mathcal{E}^l$, and #$\mathcal{E}^g$ denote the number of hyperedges in the hypergraphs $\mathcal{H}^a$, $\mathcal{H}^l$, and $\mathcal{H}^g$, which are derived from these datasets.

6.4 Experiment

6.4.1 Experimental Setup

We conduct our evaluation of `HyperGCL` using five diverse datasets, which include citation networks and social networks [146]. Table 6.1 offers detailed statistics for these datasets, along with constructed hypergraph statistics. The global node counts for the Cora, Citeseer, Wiki, Twitch-PT (PT), and LastFMAsia (LFMA) datasets are set at 3, 1, 4, 5, and 4, respectively. We employ the community detection algorithm as described by [137] with its default parameters. Our experimental protocol adheres to a standard data split: 10% for training, 10% for validation, and 80% for testing.

To evaluate performance, we compare `HyperGCL` with sixteen baseline models, applying their default configurations. These models are grouped by their architecture: graph-based models for standard graphs and hypergraph-based models for hypergraphs. The baseline hypergraph-based models construct hypergraphs according to the methodologies in their original publications. The graph-based models include GCN [75], GAT [121], GraphSage [76], DGI [114], GMI [64], MVGRL [65], GraphCL [69], and GraphMAE [147]. Hypergraph-based models encompass HGNN [7], HCHA [50], HyperGCN [79], DHGNN [8], HNHN [140], UniGCNII [82], AllSetTransformer [83], and DHKH [143].

In the node structure encoding methods outlined in Section 6.3.3, we utilize three different encoding functions to capture various aspects of the input graph and constructed hypergraph. For integrating local connectivity information with lce, a two-layer GCN implemented in the Deep Graph Library (DGL) [144] is used. To compute ce and de, we develop learnable encoding functions using PyTorch's learnable *Embedding layer*. Since $\mathcal{H}^l$ is constructed focusing on the nodes' local structure, lce is not considered in `SHyGAN` when applied to $\mathcal{H}^l$.

We conduct ten experiments using different random splits for each run. Both node and hyperedge features are one-hot encoded, and a single-layer `HyperGCL` model optimized with Adam is utilized. Optimal hyperparameters are determined through a grid search on the validation set. The chosen hyperparameters include a learning rate of 0.001 and a dropout rate of 0.1 for regularization. For the k-NN and k-means algorithms, k values of 50 and 60 are selected, respectively. The hyperparameter s discussed in Section 6.3.1.1 is set to 2. The hyperparameters τ and θ from Section 6.3.2 are set to 0.2 and 0.8, respectively. The hyperparameter 'a' in Section 6.3.4 is set to 25, and the temperature parameter η is set to 0.5. We employ the LeakyReLU activation function, and the model is configured with two attention heads. Training is conducted over 500 epochs, with early stopping applied if validation accuracy does not improve for 100 consecutive epochs. The hidden dimension for `HyGAN` and `SHyGAN` is set to 64. We implement `HyperGCL` by using DGL with PyTorch. Our experiments are performed on an NVIDIA L40S-46GB GPU.

6.4.2 Result

We evaluate the performance of our model by conducting experiments on five distinct datasets and comparing the results with sixteen state-of-the-art baselines. The baselines are considered if their experimental results or codes are available. The outcomes, presented in Table 6.2, demonstrate the superiority of our model across all datasets.

Specifically, our model excels on the Cora dataset, achieving an impressive accuracy of

Method	Model	Cora	Citeseer	Wiki	PT	LFMA
Graph-based	GCN	80.88	67.65	60.66	65.85	80.23
	GAT	81.08	68.32	61.79	66.30	82.21
	GraphSage	80.64	69.28	60.17	63.35	79.66
	DGI	81.70	71.50	64.89	66.82	83.17
	GMI	82.70	73.0	66.12	66.98	83.55
	MVGRL	82.90	72.60	66.78	67.18	84.65
	GraphCL	82.50	72.80	67.32	67.58	83.28
	GraphMAE	83.80	72.40	67.93	67.92	84.01
Hypergraph-based	HGNN	71.31	65.12	65.24	66.41	78.26
	HCHA	71.41	65.43	64.41	63.52	79.44
	HyperGCN	60.96	53.20	65.84	62.44	77.89
	DHGNN	72.22	64.59	65.87	65.37	77.22
	HNHN	65.76	63.93	63.92	66.12	81.17
	UniGCNII	70.20	65.57	66.25	64.24	80.49
	AllSetTransformer	70.99	66.60	67.44	65.15	82.42
	DHKH	64.21	66.34	66.50	67.04	80.25
	HyperGCL$_{sim}$	**84.38**	**71.35**	**68.11**	**68.88**	**84.12**
	HyperGCL$_{dis}$	**85.88**	**73.12**	**69.22**	**70.10**	**85.15**

Table 6.2 Performance Comparisons: Mean accuracy (%)

85.88%. This significantly surpasses the accuracy of the best-performing graph-based baseline model, GraphMAE, at 83.80% and exceeds the top-reported accuracy of the hypergraph-based baseline, DHGNN, which stands at 72.22%. In the case of the Citeseer dataset, our model attains an accuracy of 73.12%, outperforming the graph-based leading baseline GraphCL with an accuracy of 72.80%, and the hypergraph-based top-performing baseline, AllSetTransformer, at 66.60%. The trend continues with the Wiki, Twitch-PT, and LastFMAsia datasets, where our model substantially outperforms the baselines. The results underscore our model's substantial enhancements in classifying the datasets, setting a new standard compared to existing state-of-the-art methods. Moreover, this table shows that our model with distance-based negative sampling performs better than similarity-based negative sampling. Distance-based negative sampling chooses negative samples for a node based on network connectivity information, whereas similarity-based negative sampling uses node feature information to choose negative samples. Thus, based on the performance, we can infer that network connectivity information is more important.

A closer look at Table 6.2 reveals that hypergraph-based models generally lag behind the top-performing graph-based models. Traditional HyperGNNs are effective at capturing higher-order global structural information from the data. However, they might miss some important local structural information as they do not consider local connection details. Additionally, the baseline models typically create hypergraphs based on a single aspect of the underlying data. In contrast, our approach generates different types of hypergraphs by leveraging multiple aspects of the input data. Nonetheless, hypergraph-based models like DHGNN, AllSetTransformer, and DHKH show better performance compared to other hypergraph-based models. Specifically, DHGNN and DHKH simultaneously learn the hypergraph structure and hypergraph neural network, enabling them to prune noisy and task-irrelevant connections, thus improving performance. The AllSetTransformer framework, which blends Deep Sets and Set Transformers with hypergraph neural networks, offers substantial modeling flexibility and expressive power, enhancing performance in various tasks.

6.4.2.1 Case Study: Impact of Hypergraph Views

In our model, we generate three different hypergraph views expecting that each of them captures different aspects of information. To understand their impact, we consider running $\texttt{HyperGCL}_{dis}$ without (W/O) considering a specific hypergraph view at a time. The results are presented in Table 6.3, illustrating the influence of each individual hypergraph view on the model's performance. The data shows that the most significant performance degradation occurs when we remove the global structure-infused hypergraph view $\mathcal{H}^g$.

HyperGCLw/o	**Cora**	**Citeseer**	**Wiki**	**PT**	**LFMA**
$\mathcal{H}^g$	82.65	70.84	66.89	68.23	82.42
$\mathcal{H}^l$	83.78	71.05	67.92	67.25	83.11
$\mathcal{H}^a$	83.15	71.23	67.74	68.11	81.98
Augmentation	83.88	72.36	67.52	67.78	83.20
SHyGAN	84.05	72.28	68.49	68.66	84.29
NetCL	82.45	72.03	67.88	68.17	83.89
HyperGCL$_{dis}$	85.88	73.12	69.22	70.10	85.15

Table 6.3 Impact of different components of HyperGCL on the model performance (accuracy).

6.4.2.2 Case Study: Impact of Adaptive View Augmentation

After creating hypergraph views, our model applies a learnable Gumbel-Softmax function to adaptively augment each view. This process is intended to generate robust samples for contrastive learning, enhancing the diversity of training examples. It also refines the constructed views by selectively highlighting important relationships within the hypergraph. To assess its impact on the model's overall performance, we remove the adaptive augmentation component and compare the outputs with those of the main model across all datasets. The results, shown in Table 6.3, indicate that removing this component leads to a noticeable decline in the model's performance.

6.4.2.3 Case Study: Impact of SHyGAN

To learn the view representations, we utilize view-specific encoders. We use HyGAN to $\mathcal{H}^a$ and $\mathcal{H}^l$ and present a special variant of HyGAN named SHyGAN to $\mathcal{H}^g$. It incorporates learnable node structure encodings to improve the node representations. Moreover, employ a topology-guided attention network to discover the important nodes and hyperedges from both semantic and structural viewpoints. To evaluate the effect of SHyGAN we perform an experiment where we replace HyGAN with SHyGAN. The results in Table 6.3 demonstrate its impact.

6.4.2.4 Case study- Impact of `NetCL`

Traditional GCL methods often use contrastive loss from computer vision, treating an anchor node and its representations across views as positive samples, while considering all other nodes as negative samples. This approach overlooks the network's structural information. Our approach defines `NetCL` to incorporate network connectivity information when defining positive and negative samples. To evaluate the effectiveness of `NetCL`, we conduct an experiment without it, treating the anchor node and its representations across views as positive samples and all others as negative samples. The results, detailed in Table 6.3, demonstrate that removing `NetCL` results in a reduction in the model's overall performance.

CHAPTER 7
FUTURE WORK

The research presented in this dissertation has made significant strides in developing hypergraph neural networks (HyperGNNs) for drug-drug interaction prediction, sequence classification, structure-aware hypergraph transformers, and graph contrastive learning from hypergraph viewpoints. However, several promising avenues for future research can build upon these contributions.

- **Enhanced Hypergraph Representations** While the current work leverages HyperGNNs to capture higher-order relationships in data, future research could explore more sophisticated hypergraph construction techniques. This includes dynamically adapting the hypergraph structure based on the evolving data context or incorporating additional domain-specific knowledge to improve representation quality.

- **Scalability and Efficiency Improvements** The models proposed in this dissertation, such as `SaHT`, `HyperGCL` demonstrate state-of-the-art performance but often come with high computational costs. Future work should focus on optimizing these models for better scalability and efficiency. Techniques like model pruning, quantization, and the development of more efficient training algorithms (e.g., distributed training) could be investigated to reduce resource consumption without compromising performance.

- **Integration with Large Language Models (LLMs)** Given the recent advancements in LLMs, integrating these models with HyperGNNs could open new possibilities for enhanced learning and inference. Future research can explore hybrid architectures that leverage the strengths of both LLMs and HyperGNNs to improve tasks such as node classification, link prediction, and graph generation.

- **Explainability and Interpretability** One of the ongoing challenges in deep learning, including HyperGNNs, is the interpretability of the models. Future work should prioritize enhancing the explainability of these models, making their predictions more transparent and understandable to end-users. This could involve developing novel visualization tools, interpretability algorithms, and integrating explainable AI (XAI) techniques into the hypergraph learning framework.

- **Robustness and Adversarial Resistance** Ensuring the robustness of HyperGNNs against adversarial attacks is essential for their deployment in critical applications. Future research should focus on identifying vulnerabilities in the current models and developing robust training methods that can withstand adversarial manipulations. This includes investigating adversarial training techniques specifically tailored for hypergraph structures.

- **Hybrid and Multi-modal Approaches** The integration of HyperGNNs with other machine learning paradigms, such as reinforcement learning, transfer learning, and multi-modal learning, represents a fertile ground for future research. Exploring how these combined approaches can enhance performance and expand the applicability of hypergraph-based models will be a key area of interest.

- **Continuous Learning and Adaptation** In dynamic environments, it is crucial for models to adapt continuously to new data and evolving patterns. Future work could explore continuous learning frameworks for HyperGNNs, enabling them to learn incrementally from new data without forgetting previously acquired knowledge.

By addressing these future directions, the field of HyperGNNs and their applications can continue to evolve, pushing the boundaries of what is possible in graph learning and its intersection with other areas of artificial intelligence.

CHAPTER 8
CONCLUSION

In this dissertation, we have explored various facets of hypergraph neural networks (HyperGNNs) and their applications across different domains. Through four distinct yet interconnected research projects, we have advanced the state of the art in hypergraph learning, emphasizing the importance of HyperGNNs in graph mining, bioinformatics, and network science. The key contributions and findings from each paper are summarized below.

`HyGNN`: In the first project, we introduced `HyGNN` a novel model designed for drug-drug interaction (DDI) prediction. By constructing a drug hypergraph from SMILES strings and modeling a novel HyperGNN, we were able to depict complex similarities between chemical structures and predict DDIs with higher accuracy than traditional methods.

`Seq-HyGAN`: In the first project, we introduced `Seq-HyGAN` which leverages an attention network within hypergraph for sequence classification, capturing higher-order structural similarities among sequences. The proposed three-level attention model effectively learns hyperedge representations, leading to superior performance on various biological datasets.

`SaHT`: In the third project, we presented a transformer network for hypergraphs named `SaHT` that incorporates both semantic and topological information for node classification. By integrating structural and spatial encoding with a self-attention mechanism, SaHT significantly outperforms existing graph and hypergraph transformer models, showcasing its potential to revolutionize hypergraph analytics.

`HyperGCL`: In the fourth project, we developed `HyperGCL` a graph contrastive learning framework that utilizes three distinct hypergraph views to capture comprehensive attribute and structural information. Through a learnable Gumbel-Softmax function, view-specific encoders, and a network-aware contrastive loss, `HyperGCL` addresses critical limitations in existing methods, achieving state-of-the-art performance in node classification tasks.

These projects collectively demonstrate the versatility and effectiveness of HyperGNNs in addressing complex problems across various domains. Our research advances the capabilities of HyperGNNs, showcasing their superior performance compared to existing methods in diverse applications. The advancements presented in this dissertation establish a solid foundation for further exploration and innovation in hypergraph neural networks, paving the way for the development of more robust, interpretable, and practical models in the future.

REFERENCES

[1] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*, pages 2535–2546, 2021.

[2] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022.

[3] Rui Li, Xin Yuan, Mohsen Radfar, Peter Marendy, Wei Ni, Terrence J O'Brien, and Pablo M Casillas-Espinosa. Graph signal processing, graph neural network and graph learning on biological data: a systematic review. *IEEE Reviews in Biomedical Engineering*, 16:109–135, 2021.

[4] Alessia Antelmi, Gennaro Cordasco, Mirko Polato, Vittorio Scarano, Carmine Spagnuolo, and Dingqi Yang. A survey on hypergraph representation learning. *ACM Computing Surveys*, 56(1):1–38, 2023.

[5] Hyunjin Choo and Kijung Shin. On the persistence of higher-order interactions in real-world hypergraphs. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 163–171. SIAM, 2022.

[6] Alain Bretto. Hypergraph theory. *An introduction. Mathematical Engineering. Cham: Springer*, 1, 2013.

[7] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, vol-

ume 33, pages 3558–3565, 2019.

[8] Jianwen Jiang, Yuxuan Wei, Yifan Feng, Jingxuan Cao, and Yue Gao. Dynamic hypergraph neural networks. In *IJCAI*, pages 2635–2641, 2019.

[9] Chaofan Chen, Zelei Cheng, Zuotian Li, and Manyi Wang. Hypergraph attention networks. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1560–1565. IEEE, 2020.

[10] Kaize Ding, Jianling Wang, Jundong Li, Dingcheng Li, and Huan Liu. Be more with less: Hypergraph attention networks for inductive text classification. *arXiv preprint arXiv:2011.00387*, 2020.

[11] Yi Jiang, Ruheng Wang, Jiuxin Feng, Junru Jin, Sirui Liang, Zhongshen Li, Yingying Yu, Anjun Ma, Ran Su, Quan Zou, et al. Explainable deep hypergraph learning modeling the peptide secondary structure prediction. *Advanced Science*, 10(11):2206151, 2023.

[12] Shanchen Pang, Kuijie Zhang, Shudong Wang, Yuanyuan Zhang, Sicheng He, Wenhao Wu, and Sibo Qiao. Hgdd: A drug-disease high-order association information extraction method for drug repurposing via hypergraph. In *International symposium on bioinformatics research and applications*, pages 424–435. Springer, 2021.

[13] Ke Han, Peigang Cao, Yu Wang, Fang Xie, Jiaqi Ma, Mengyao Yu, Jianchun Wang, Yaoqun Xu, Yu Zhang, and Jie Wan. A review of approaches for predicting drug–drug interactions based on machine learning. *Frontiers in pharmacology*, 12:814858, 2022.

[14] Yanchao Tan, Chengjun Kong, Leisheng Yu, Pan Li, Chaochao Chen, Xiaolin Zheng,

Vicki S Hertzberg, and Carl Yang. 4sdrug: Symptom-based set-to-set small and safe drug recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3970–3980, 2022.

[15] Yang Qiu, Yang Zhang, Yifan Deng, Shichao Liu, and Wen Zhang. A comprehensive review of computational methods for drug-drug interaction detection. *IEEE/ACM transactions on computational biology and bioinformatics*, 19(4):1968–1985, 2021.

[16] Hai-Cheng Yi, Zhu-Hong You, De-Shuang Huang, and Chee Keong Kwoh. Graph representation learning in bioinformatics: trends, methods and applications. *Briefings in Bioinformatics*, 23(1):bbab340, 2022.

[17] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.

[18] Takako Takeda, Ming Hao, Tiejun Cheng, Stephen H Bryant, and Yanli Wang. Predicting drug–drug interactions through drug structural similarities and interaction networks incorporating pharmacokinetics and pharmacodynamics knowledge. *Journal of cheminformatics*, 9:1–9, 2017.

[19] Yi Zheng, Hui Peng, Xiaocai Zhang, Zhixun Zhao, Xiaoying Gao, and Jinyan Li. Ddi-pulearn: a positive-unlabeled learning method for large-scale prediction of drug-drug interactions. *BMC bioinformatics*, 20:1–12, 2019.

[20] Farhan Tanvir, Muhammad Ifte Khairul Islam, and Esra Akbas. Predicting drug-drug interactions using meta-path based similarities. In *2021 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*,

pages 1–8. IEEE, 2021.

[21] Yue-Hua Feng, Shao-Wu Zhang, and Jian-Yu Shi. Dpddi: a deep predictor for drug-drug interactions. *BMC bioinformatics*, 21(1):419, 2020.

[22] Zhong-Hao Ren, Chang-Qing Yu, Li-Ping Li, Zhu-Hong You, Yong-Jian Guan, Xin-Fei Wang, and Jie Pan. Biodkg–ddi: predicting drug–drug interactions based on drug knowledge graph fusing biochemical information. *Briefings in Functional Genomics*, 21(3):216–229, 2022.

[23] Yue Yu, Kexin Huang, Chao Zhang, Lucas M Glass, Jimeng Sun, and Cao Xiao. Sumgnn: multi-typed drug interaction prediction via efficient knowledge graph summarization. *Bioinformatics*, 37(18):2988–2995, 2021.

[24] Suyu Mei and Kun Zhang. A machine learning framework for predicting drug–drug interactions. *Scientific Reports*, 11(1):17619, 2021.

[25] Santiago Vilar, Rave Harpaz, Eugenio Uriarte, Lourdes Santana, Raul Rabadan, and Carol Friedman. Drug—drug interaction through molecular structure similarity analysis. *Journal of the American Medical Informatics Association*, 19(6):1066–1074, 2012.

[26] Yvonne C Martin, James L Kofron, and Linda M Traphagen. Do structurally similar molecules have similar biological activity? *Journal of medicinal chemistry*, 45(19):4350–4358, 2002.

[27] Ngoc G Nguyen, Vu Anh Tran, Dau Phan, Favorisen R Lumbanraja, Mohammad Reza Faisal, Bahriddin Abapihi, Mamoru Kubo, and Kenji Satou. Dna sequence classification by convolutional neural network. *Journal Biomedical Science and Engineering*,

9(5):280–286, 2016.

[28] Mirko Torrisi, Gianluca Pollastri, and Quan Le. Deep learning methods in protein structure prediction. *Computational and Structural Biotechnology Journal*, 18:1301–1310, 2020.

[29] Patrick Brendan Timmons and Chandralal M Hewage. Ennaact is a novel tool which employs neural networks for anticancer activity classification for therapeutic peptides. *Biomedicine & Pharmacotherapy*, 133:111051, 2021.

[30] Khaled Mohammed Saifuddin, Muhammad Ifte Khairul Islam, and Esra Akbas. Drug abuse detection in twitter-sphere: Graph-based approach. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 4136–4145. IEEE, 2021.

[31] Yuhang Guo, Xiao Luo, Liang Chen, and Minghua Deng. Dna-gcn: Graph convolutional networks for predicting dna-protein binding. In *International conference on intelligent computing*, pages 458–466. Springer, 2021.

[32] Vladimir Gligorijević, P Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Daniel Berenberg, Tommi Vatanen, Chris Chandler, Bryn C Taylor, Ian M Fisk, Hera Vlamakis, et al. Structure-based protein function prediction using graph convolutional networks. *Nature communications*, 12(1):3168, 2021.

[33] Bri Bumgardner, Farhan Tanvir, Khaled Mohammed Saifuddin, and Esra Akbas. Drug-drug interaction prediction: a purely smiles based approach. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5571–5579. IEEE, 2021.

[34] Yujie Qian. *A graph-based framework for information extraction.* PhD thesis, Mas-

sachusetts Institute of Technology, 2019.

[35] Muhammad Ifte Islam, Farhan Tanvir, Ginger Johnson, Esra Akbas, and Mehmet Emin Aktas. Proximity-based compression for network embedding. *Frontiers in big Data*, 3:608043, 2021.

[36] Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. Why are de bruijn graphs useful for genome assembly? *Nature biotechnology*, 29(11):987, 2011.

[37] Holly J Atkinson, John H Morris, Thomas E Ferrin, and Patricia C Babbitt. Using sequence similarity networks for visualization of relationships across diverse protein superfamilies. *PloS one*, 4(2):e4345, 2009.

[38] Jaroslaw Zola. Constructing similarity graphs from large-scale biological sequence collections. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 500–507. IEEE, 2014.

[39] Mehmet Emin Aktas, Thu Nguyen, Sidra Jawaid, Rakin Riza, and Esra Akbas. Identifying critical higher-order interactions in complex networks. *Scientific reports*, 11(1):21288, 2021.

[40] Mehmet Emin Aktas and Esra Akbas. Hypergraph laplacians in diffusion framework. In *Complex Networks & Their Applications X: Volume 2, Proceedings of the Tenth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2021 10*, pages 277–288. Springer, 2022.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in*

neural information processing systems, 30, 2017.

[42] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

[43] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: A tokenized graph transformer for node classification in large graphs. In *The Eleventh International Conference on Learning Representations*, 2022.

[44] Zaixi Zhang, Qi Liu, Qingyong Hu, and Chee-Kong Lee. Hierarchical graph transformer with adaptive node sampling. *Advances in Neural Information Processing Systems*, 35:21171–21183, 2022.

[45] Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35:27387–27401, 2022.

[46] Wenhao Zhu, Tianyu Wen, Guojie Song, Liang Wang, and Bo Zheng. On structural expressive power of graph transformers. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3628–3637, New York, NY, USA, 2023. Association for Computing Machinery.

[47] Han Li, Dan Zhao, and Jianyang Zeng. Kpgt: knowledge-guided pre-training of graph transformer for molecular property prediction. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 857–867, 2022.

[48] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim.

Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[49] Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. Gophormer: Ego-graph transformer for node classification. *arXiv preprint arXiv:2110.13094*, 2021.

[50] Song Bai, Feihu Zhang, and Philip HS Torr. Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110:107637, 2021.

[51] Mengran Li, Yong Zhang, Xiaoyong Li, Yuchen Zhang, and Baocai Yin. Hypergraph transformer neural networks. *ACM Transactions on Knowledge Discovery from Data*, 17(5):1–22, 2023.

[52] Lianghao Xia, Chao Huang, and Chuxu Zhang. Self-supervised hypergraph transformer for recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2100–2109, 2022.

[53] Haopeng Zhang, Xiao Liu, and Jiawei Zhang. Hegel: Hypergraph transformer for long document summarization. *arXiv preprint arXiv:2210.04126*, 2022.

[54] Kaize Ding, Albert Jiongqian Liang, Bryan Perozzi, Ting Chen, Ruoxi Wang, Lichan Hong, Ed H Chi, Huan Liu, and Derek Zhiyuan Cheng. Hyperformer: Learning expressive sparse feature representations via hypergraph transformer. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2062–2066, 2023.

[55] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple

framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[56] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[57] Yangguang Li, Feng Liang, Lichen Zhao, Yufeng Cui, Wanli Ouyang, Jing Shao, Fengwei Yu, and Junjie Yan. Supervision exists everywhere: A data efficient contrastive language-image pre-training paradigm. In *International Conference on Learning Representations*, 2022.

[58] Jiangmeng Li, Wenwen Qiang, Changwen Zheng, Bing Su, and Hui Xiong. Metaug: Contrastive learning via meta feature augmentation. In *International Conference on Machine Learning*, pages 12964–12978. PMLR, 2022.

[59] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 776–794. Springer, 2020.

[60] Haiyan Wu, Yanyun Qu, Shaohui Lin, Jian Zhou, Ruizhi Qiao, Zhizhong Zhang, Yuan Xie, and Lizhuang Ma. Contrastive learning for compact single image dehazing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10551–10560, 2021.

[61] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*, 2021.

[62] Hongchao Fang, Sicheng Wang, Meng Zhou, Jiayuan Ding, and Pengtao Xie. Cert: Contrastive self-supervised learning for language understanding. *arXiv preprint arXiv:2005.12766*, 2020.

[63] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2019.

[64] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*, pages 259–270, 2020.

[65] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International conference on machine learning*, pages 4116–4126. PMLR, 2020.

[66] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems*, 34:15920–15933, 2021.

[67] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, 2021.

[68] Xiao Shen, Dewang Sun, Shirui Pan, Xi Zhou, and Laurence T Yang. Neighbor contrastive learning on learnable graph augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9782–9791, 2023.

[69] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

[70] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L Dyer, Remi Munos, Petar Veličković, and Michal Valko. Large-scale representation learning on graphs via bootstrapping. *arXiv preprint arXiv:2102.06514*, 2021.

[71] Jiaqi Zeng and Pengtao Xie. Contrastive self-supervised learning for graph classification. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 35, pages 10824–10832, 2021.

[72] Jialu Chen and Gang Kou. Attribute and structure preserving graph contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7024–7032, 2023.

[73] Hyunjin Hwang, Seungwoo Lee, and Kijung Shin. Hyfer: A framework for making hypergraph learning easy, scalable and benchmarkable. In *WWW Workshop on Graph Learning Benchmarks*, 2021.

[74] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[75] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[76] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[77] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[78] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Message passing neural networks. *Machine learning meets quantum physics*, pages 199–214, 2020.

[79] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. *Advances in neural information processing systems*, 32, 2019.

[80] Naganand Yadati. Neural message passing for multi-relational ordered and recursive hypergraphs. *Advances in Neural Information Processing Systems*, 33:3275–3289, 2020.

[81] Devanshu Arya, Deepak K Gupta, Stevan Rudinac, and Marcel Worring. Hypersage: Generalizing inductive representation learning on hypergraphs. *arXiv preprint arXiv:2010.04558*, 2020.

[82] Jing Huang and Jie Yang. Unignn: a unified framework for graph and hypergraph neural networks. *arXiv preprint arXiv:2105.00956*, 2021.

[83] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are allset: A multiset function framework for hypergraph neural networks. *arXiv preprint arXiv:2106.13264*, 2021.

[84] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information*

processing systems, 30, 2017.

[85] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[86] Assaf Gottlieb, Gideon Y Stein, Yoram Oron, Eytan Ruppin, and Roded Sharan. Indi: a computational framework for inferring drug interactions and their associated recommendations. *Molecular systems biology*, 8(1):592, 2012.

[87] Reza Ferdousi, Reza Safdari, and Yadollah Omidi. Computational prediction of drug-drug interactions based on drugs functional similarities. *Journal of biomedical informatics*, 70:54–64, 2017.

[88] Heba Ibrahim, Ahmed M El Kerdawy, A Abdo, and A Sharaf Eldin. Similarity-based machine learning framework for predicting safety signals of adverse drug–drug interactions. *Informatics in Medicine Unlocked*, 26:100699, 2021.

[89] Behrooz Davazdahemami and Dursun Delen. A chronological pharmacovigilance network analytics approach for predicting adverse drug events. *Journal of the American Medical Informatics Association*, 25(10):1311–1321, 2018.

[90] Heng Luo, Ping Zhang, Hui Huang, Jialiang Huang, Emily Kao, Leming Shi, Lin He, and Lun Yang. Ddi-cpi, a server that predicts drug–drug interactions through implementing the chemical–protein interactome. *Nucleic acids research*, 42(W1):W46–W52, 2014.

[91] Andrej Kastrin, Polonca Ferk, and Brane Leskošek. Predicting potential drug-drug interactions on topological and semantic similarity features using statistical learning. *PloS one*, 13(5):e0196865, 2018.

[92] Farhan Tanvir, Khaled Mohammed Saifuddin, and Esra Akbas. Ddi prediction via heterogeneous graph attention networks. *arXiv preprint arXiv:2207.05672*, 2022.

[93] Xin Chen, Xien Liu, and Ji Wu. Drug-drug interaction prediction with graph representation learning. In *2019 IEEE International conference on bioinformatics and biomedicine (BIBM)*, pages 354–361. IEEE, 2019.

[94] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. Utilizing graph machine learning within drug discovery and development. *Briefings in bioinformatics*, 22(6):bbab159, 2021.

[95] Yunsheng Bai, Ken Gu, Yizhou Sun, and Wei Wang. Bi-level graph neural networks for drug-drug interaction prediction. *arXiv preprint arXiv:2006.14002*, 2020.

[96] Kexin Huang, Cao Xiao, Trong Hoang, Lucas Glass, and Jimeng Sun. Caster: Predicting drug interactions with chemical substructure representation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 702–709, 2020.

[97] Jae Yong Ryu, Hyun Uk Kim, and Sang Yup Lee. Deep learning improves prediction of drug–drug and drug–food interactions. *Proceedings of the national academy of sciences*, 115(18):E4304–E4311, 2018.

[98] Rafael Gómez-Bombarelli, David Kristjanson Duvenaud, José Miguel Hernández-

Lobato, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4:268 – 276, 2018.

[99] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Biocomputing 2002*, pages 564–575. World Scientific, 2001.

[100] Hiroto Saigo, Jean-Philippe Vert, Nobuhisa Ueda, and Tatsuya Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.

[101] Jesse Eickholt and Jianlin Cheng. Dndisorder: predicting protein disorder using boosting and deep networks. *BMC bioinformatics*, 14:1–10, 2013.

[102] Haitham Ashoor, Xiaowen Chen, Wojciech Rosikiewicz, Jiahui Wang, Albert Cheng, Ping Wang, Yijun Ruan, and Sheng Li. Graph embedding and unsupervised learning predict genomic sub-compartments from hic chromatin interaction data. *Nature communications*, 11(1):1173, 2020.

[103] Sohyun Hwang, Chan Yeong Kim, Sunmo Yang, Eiru Kim, Traver Hart, Edward M Marcotte, and Insuk Lee. Humannet v2: human gene networks for disease research. *Nucleic acids research*, 47(D1):D573–D580, 2019.

[104] Isaac Turner, Kiran V Garimella, Zamin Iqbal, and Gil Mcvean. Integrating long-range connectivity information into de bruijn graphs.

[105] Alyssa Quek, Zhiyong Wang, Jian Zhang, and Dagan Feng. Structural image classifi-

cation with graph neural networks. In *2011 International Conference on Digital Image Computing: Techniques and Applications*, pages 416–421. IEEE, 2011.

[106] Kai Yao Huang, Yi Jhan Tseng, Hui Ju Kao, Chia Hung Chen, Hsiao Hsiang Yang, and Shun Long Weng. Identification of subtypes of anticancer peptides based on sequential features and physicochemical properties. *Scientific Reports 2021 11:1*, 11:1–13, 6 2021.

[107] Yufeng Zhang, Xueli Yu, Zeyu Cui, Shu Wu, Zhongzhen Wen, and Liang Wang. Every document owns its structure: Inductive text classification via graph neural networks.

[108] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.

[109] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[110] Jie Liu, Lingyun Song, Guangtao Wang, and Xuequn Shang. Meta-hgt: Metapath-aware hypergraph transformer for heterogeneous information network embedding. *Neural Networks*, 157:65–76, 2023.

[111] Haopeng Zhang, Xiao Liu, and Jiawei Zhang. HEGEL: Hypergraph transformer for long document summarization. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10167–10176, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[112] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond,

Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[113] John Giorgi, Osvald Nitski, Bo Wang, and Gary Bader. Declutr: Deep contrastive learning for unsupervised textual representations. *arXiv preprint arXiv:2006.03659*, 2020.

[114] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[115] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[116] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[117] Yixuan Ma and Kun Zhan. Self-contrastive graph diffusion network. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 3857–3865, 2023.

[118] Kexin Huang, Cao Xiao, Lucas Glass, and Jimeng Sun. Explainable substructure partition fingerprint for protein, drug, and more. In *NeurIPS Learning Meaningful Representation of Life Workshop*, 2019.

[119] Jingsong Zhang, Jianmei Guo, Xiaoqing Yu, Xiangtian Yu, Weifeng Guo, Tao Zeng, and Luonan Chen. Mining k-mers of various lengths in biological sequences. In *Bioin-*

formatics Research and Applications: 13th International Symposium, ISBRA 2017, Honolulu, HI, USA, May 29–June 2, 2017, Proceedings 13, pages 186–195. Springer, 2017.

[120] Kristoffer Sahlin. Strobemers: an alternative to k-mers for sequence comparison. *bioRxiv*, pages 2021–01, 2021.

[121] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[122] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[123] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[124] Nagesh Singh Chauhan. Demystify dna sequencing with machine learning — kaggle.

[125] Mingfeng Xie, Dijia Liu, and Yufeng Yang. Anti-cancer peptides: classification, mechanism of action, reconstruction and modification. *Open biology*, 10(7):200004, 2020.

[126] Francesca Grisoni, Claudia S Neuhaus, Miyabi Hishinuma, Gisela Gabernet, Jan A Hiss, Masaaki Kotera, and Gisbert Schneider. De novo design of anticancer peptides by ensemble artificial neural networks. *Journal of Molecular Modeling*, 25:1–10, 2019.

[127] UCI. Uci machine learning repository: Data sets.

[128] Sarwan Ali, Babatunde Bello, Prakash Chourasia, Ria Thazhe Punathil, Yijing Zhou,

and Murray Patterson. Pwm2vec: An efficient embedding approach for viral host specification from coronavirus spike sequences. *Biology*, 11(3):418, 2022.

[129] Sandrine Belouzard, Jean K. Millet, Beth N. Licitra, and Gary R. Whittaker. Mechanisms of coronavirus cell entry mediated by the viral spike protein. *Viruses*, 4:1011–1033, 2012.

[130] Kiril Kuzmin, Ayotomiwa Ezekiel Adeniyi, Arthur Kevin DaSouza Jr, Deuk Lim, Huyen Nguyen, Nuria Ramirez Molina, Lanqiao Xiong, Irene T Weber, and Robert W Harrison. Machine learning methods accurately predict host specificity of coronaviruses based on spike sequences alone. *Biochemical and Biophysical Research Communications*, 533(3):553–558, 2020.

[131] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[132] F Chollet. Keras.[online] available at: https://github. com/fchollet/keras. *Accessed*, 12(01):2021, 2015.

[133] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[134] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed represen-

tations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[135] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

[136] JA Rodriguez. Laplacian eigenvalues and partition problems in hypergraphs. *Applied Mathematics Letters*, 22(6):916–921, 2009.

[137] Duanbing Chen, Mingsheng Shang, Zehua Lv, and Yan Fu. Detecting overlapping communities of weighted networks via a local algorithm. *Physica A: Statistical Mechanics and its Applications*, 389(19):4177–4187, 2010.

[138] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with pytorch geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[139] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

[140] Yihe Dong, Will Sawin, and Yoshua Bengio. Hnhn: Hypergraph networks with hyperedge neurons. *arXiv preprint arXiv:2006.12278*, 2020.

[141] Iulia Duta, Giulia Cassarà, Fabrizio Silvestri, and Pietro Liò. Sheaf hypergraph networks. *Advances in Neural Information Processing Systems*, 36, 2024.

[142] Derun Cai, Moxian Song, Chenxi Sun, Baofeng Zhang, Shenda Hong, and Hongyan Li. Hypergraph structure learning for hypergraph neural networks. In *Proceedings of the*

Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, Lud De Raedt, Ed, volume 7, pages 1923–1929, 2022.

[143] Xiaojun Kang, Xinchuan Li, Hong Yao, Dan Li, Bo Jiang, Xiaoyue Peng, Tiejun Wu, Shihua Qi, and Lijun Dong. Dynamic hypergraph neural networks based on key hyperedges. *Information Sciences*, 616:37–51, 2022.

[144] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[145] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[146] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[147] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 594–604, 2022.