



UNSW
SYDNEY

Advanced Topics of Graph Neural Networks

Hanchen Wang

29/07/2024

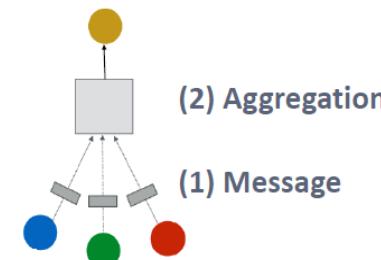
Recap: Graph Neural Networks

■ (1) Graph Convolutional Networks (GCN)

$$\mathbf{h}_v^{(l)} = \sigma \left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$

■ How to write this as Message + Aggregation?

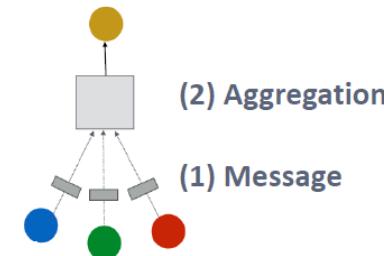
$$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \underbrace{\mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}}_{\text{Aggregation}} \underbrace{\text{Message}}_{\text{Message}} \right)$$



Recap: Graph Neural Networks

■ (1) Graph Convolutional Networks (GCN)

$$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$



■ Message:

- Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

Normalized by node degree
(In the GCN paper they use a slightly different normalization)

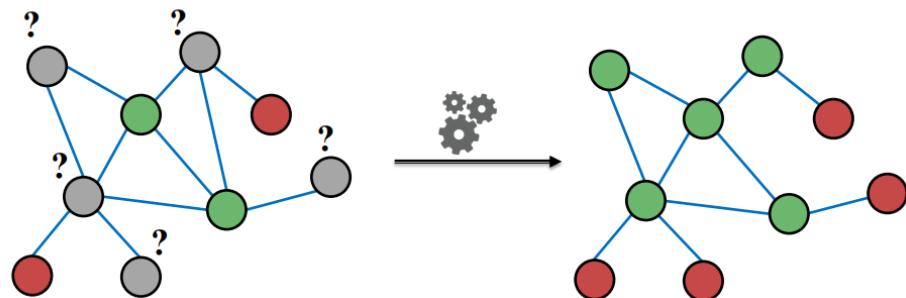
■ Aggregation:

- Sum over messages from neighbors, then apply activation

$$\mathbf{h}_v^{(l)} = \sigma \left(\text{Sum} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right) \right)$$

Application of Graph Neural Networks

■ Node Classification

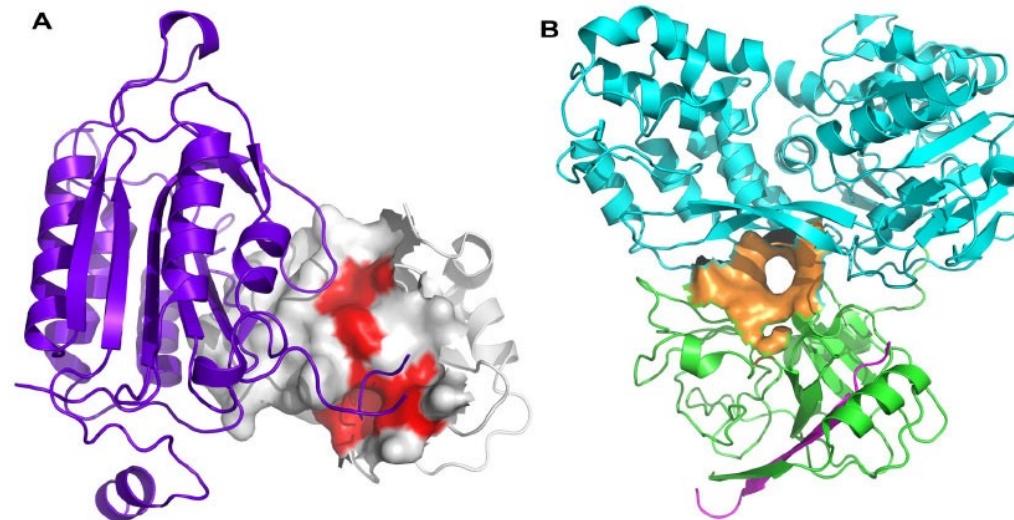
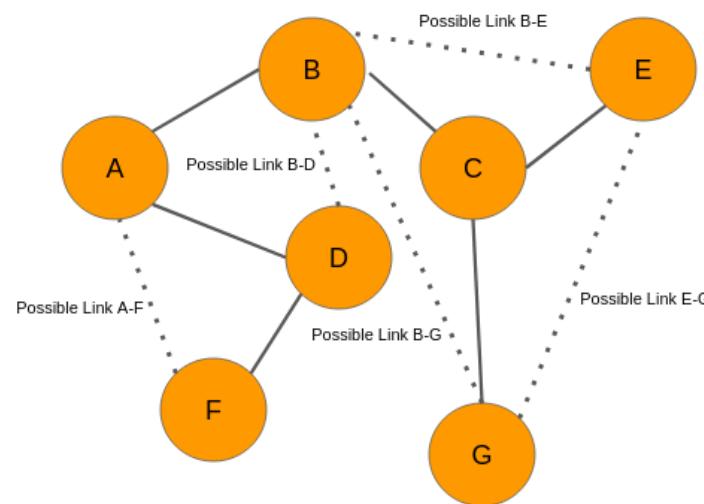


- Some labels of nodes are given
- Predict the labels of Unlabelled nodes.



Application of Graph Neural Networks

■ Link Prediction



- With existing edges (links) between nodes, predict the existence of possible links.
- Link prediction can be used to explore potential interaction between proteins.

Topics of GNNs

- Structured Entity Analysis (Chemical, Biomedical)
- Subgraph Isomorphism (Database)
- Fraud Detection (e-Commerce)
- LLM with GNNs

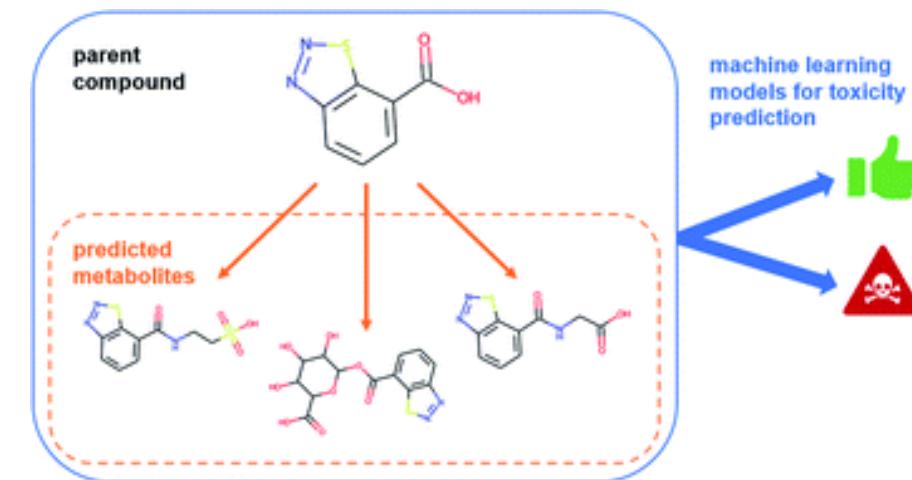
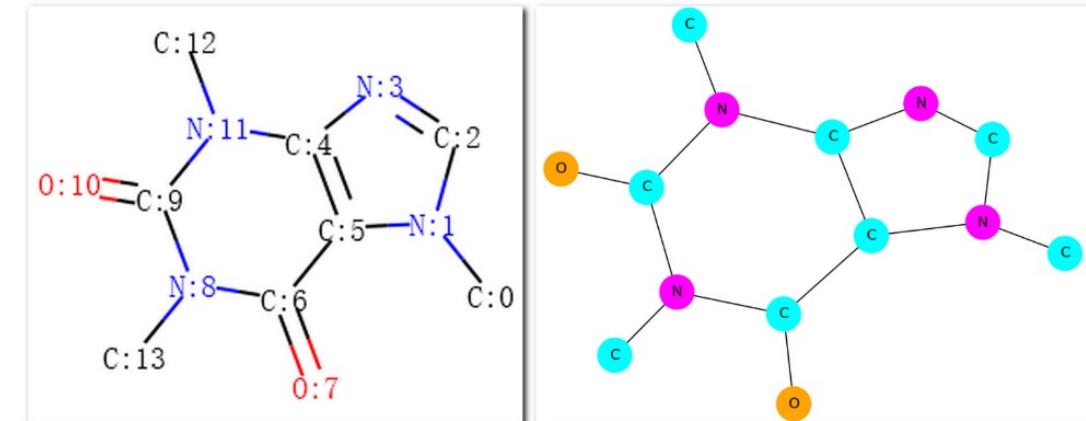


Structured Entity Analysis



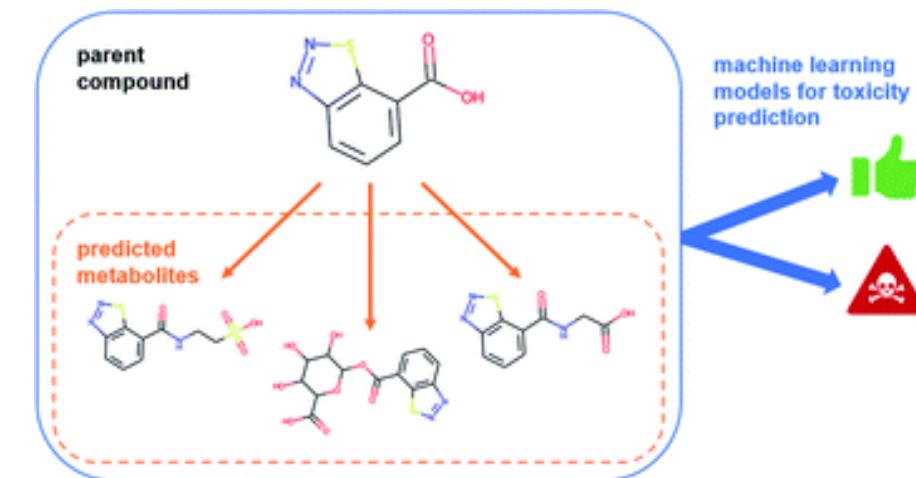
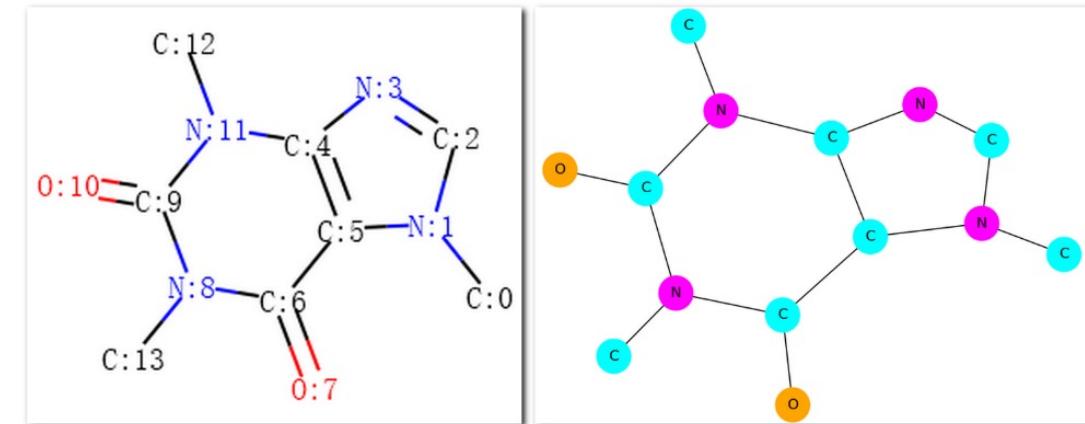
Structured Entity Analysis

- Many real-life **structured entities** can be modelled as **graphs**, such as chemical molecules and proteins.
- Graph neural networks can be used to analyse these structured entities.



Structured Entity Analysis

- Toxicity of chemical molecules.
- Property prediction for molecules.
- Interactions between molecules.



Interaction between molecules

- Prediction of reaction between two molecules
- Prediction of the side effect of drug pairs.



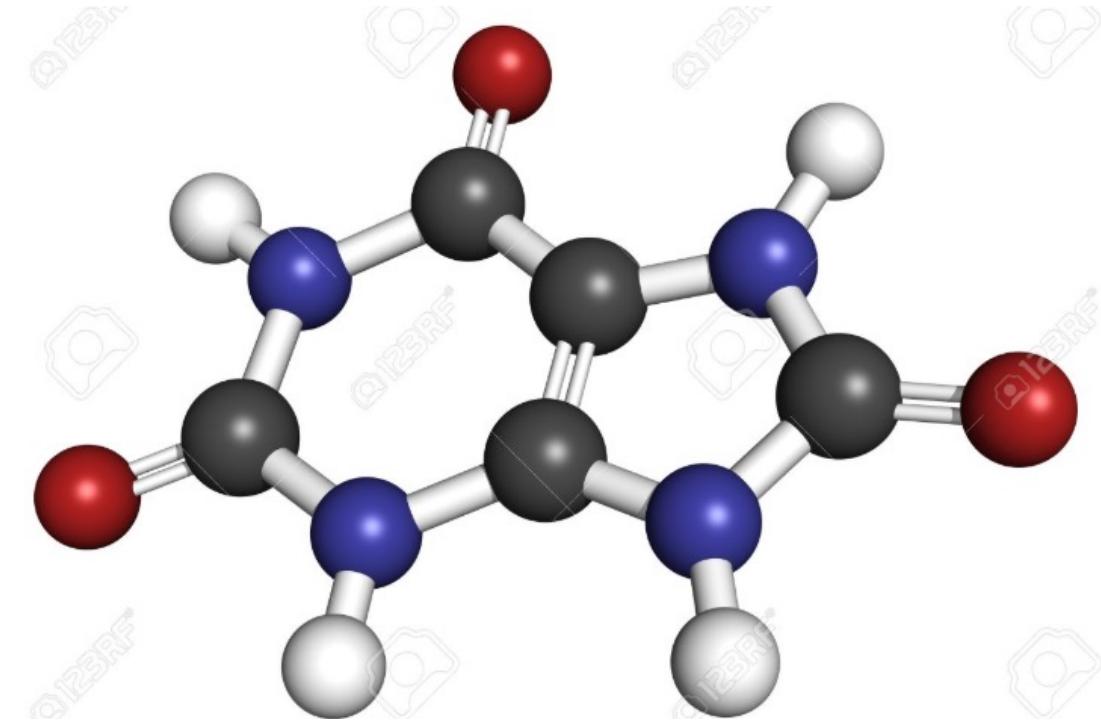
Immediate solutions

- The immediate way to investigate the interactions between two entities (molecules or drugs) is to conduct experiments in laboratories and clinics.
- However, it is time consuming and labour intensive. It is impractical to test all entity pairs.
- Therefore, computational approaches are proposed to predict structured entity interaction effectively and efficiently.



Naïve computational solution

- Model the molecules as graphs
- Directly apply the graph neural networks for the predictions.
- However, such solution ignores the **interaction relation** between molecules.

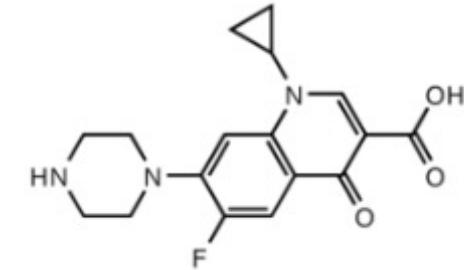


Other computational methods

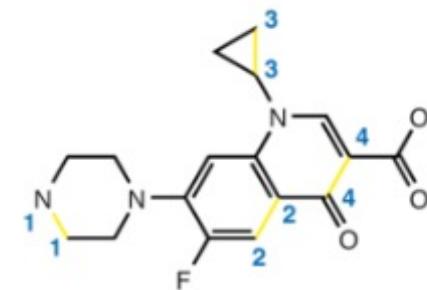
Simplified molecular-input line-entry system (SMILES)

- Using strings to represent molecules.
- Natural Language Processing (NLP) models are applied to capture the molecule information and produce the representations for molecules
- Downstream applications are based on the representations.
- However, these models cannot capture the structural information of the molecules.

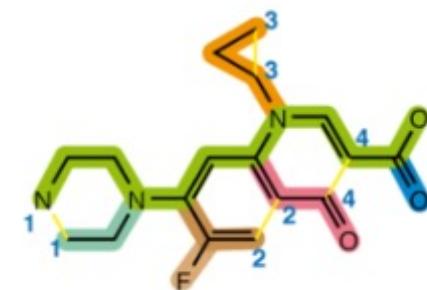
A



B



C



D

N1CCN(CC1)C(C(F)=C2)=CC(=C2C4=O)N(C3CC3)C=C4C(=O)O

W

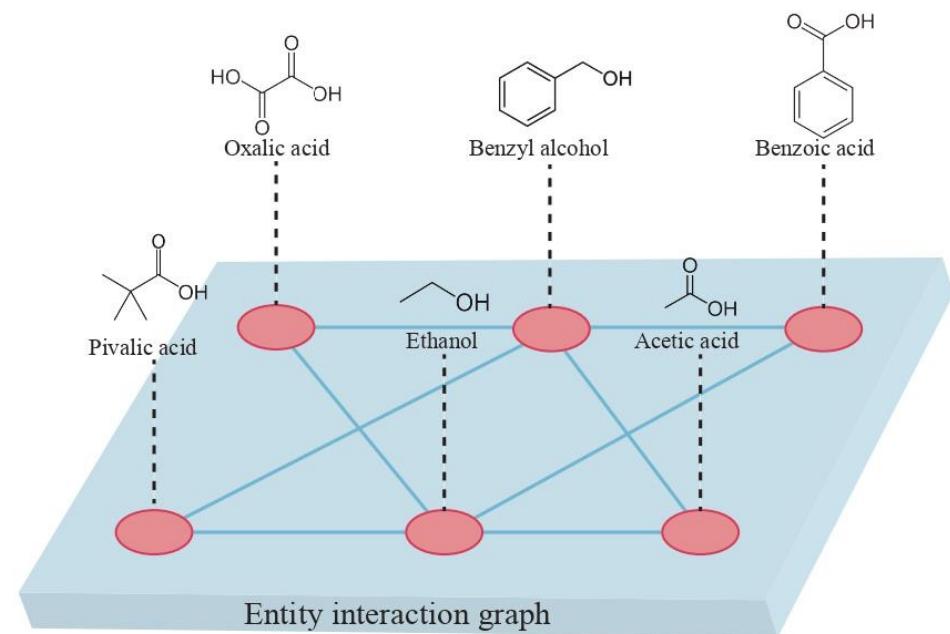
Goals

- Capture the structural information of the molecule graphs.
- Preserve the interaction relationship between the structured entities (molecules).
- Identify the important substructures within the molecules that are key to the prediction of interactions.



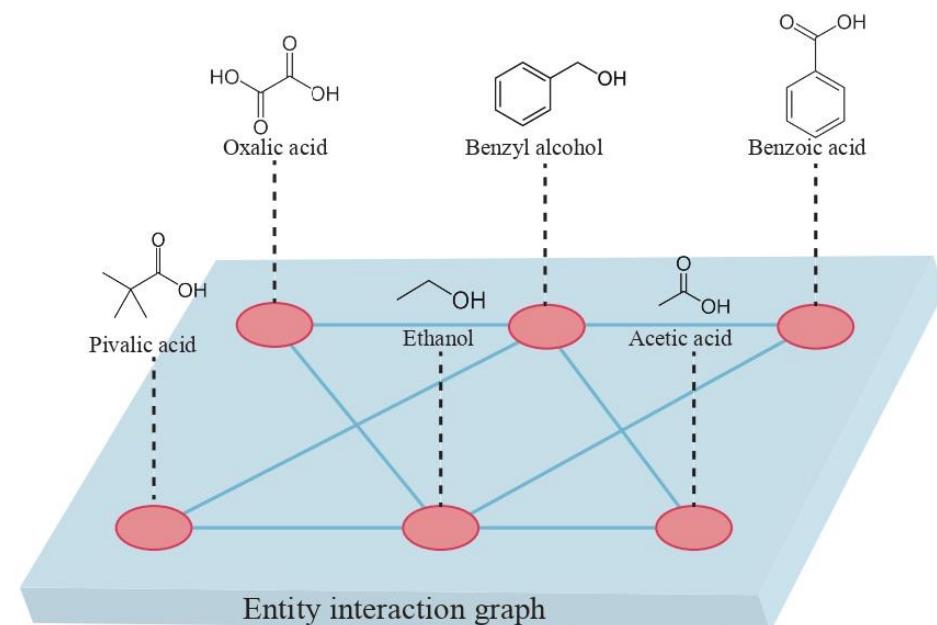
Graph of Graphs

- *Local graph:* the molecule graph representing the chemical structure.
- (nodes: atoms, edges: chemical bonds between the atoms.)
- *Global graph:* the graph of interactions between the chemicals.
- (nodes: chemical molecules, edges: interaction relations between the molecules.)



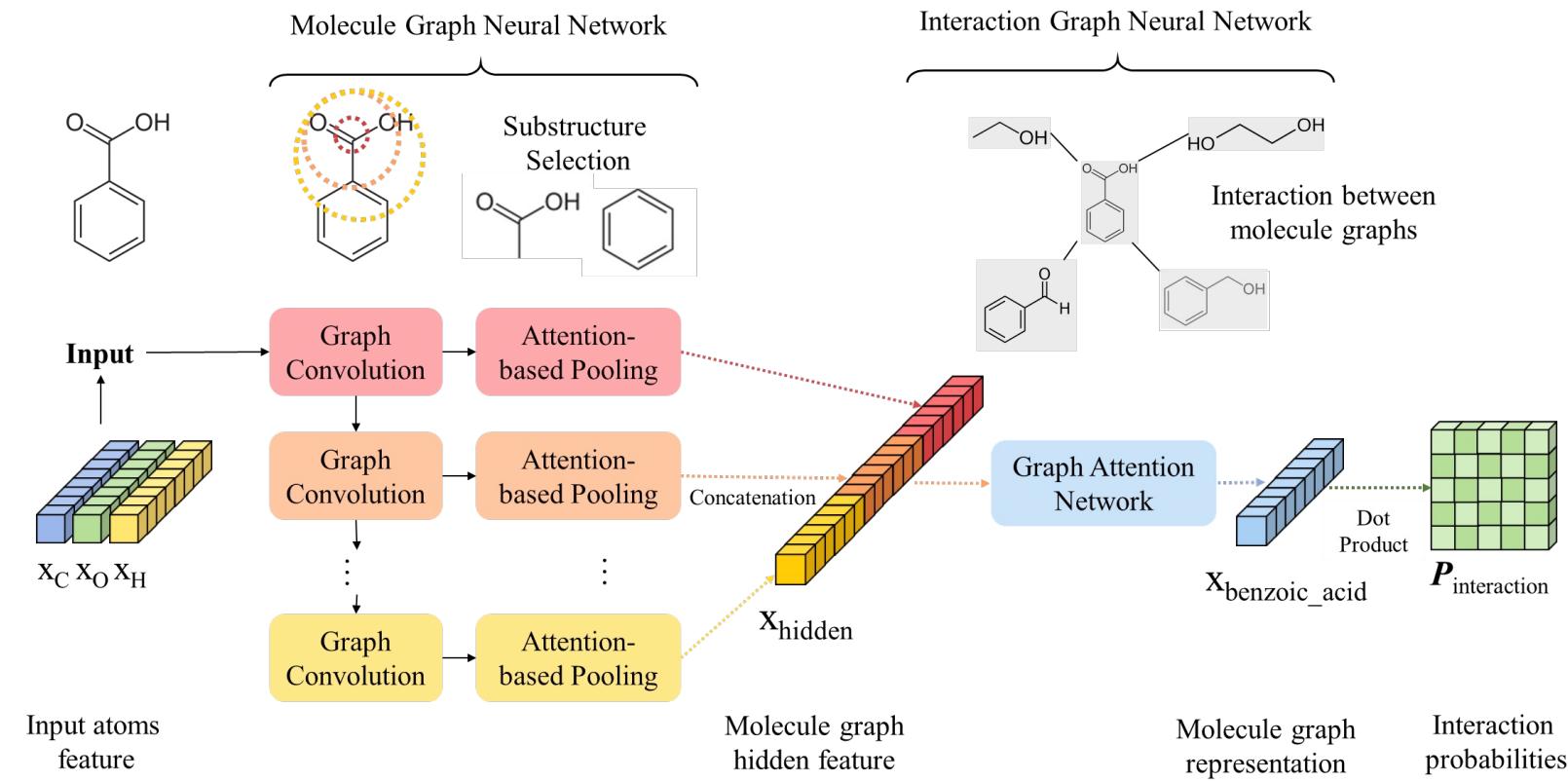
Target of the model

- With the input graph, the target of the model is to conduct the **link prediction** on the *global graph*.
- The graph neural network applied on the *local graph* should be able to capture the important substructure (functional groups) in the molecules.



Graph of Graphs Neural Network

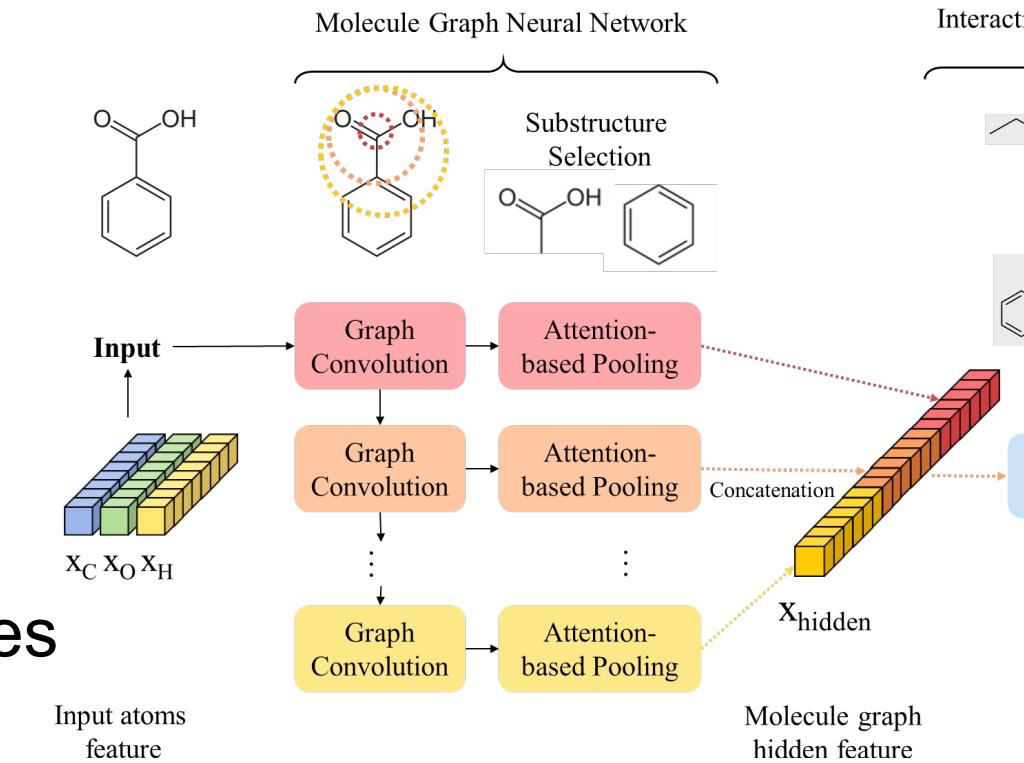
- Apply graph neural networks on the graph of graphs.
- Utilize more information within the graph structures.
- Can be used for entity interaction prediction and molecule classification.



Graph of Graphs Neural Network

Multi-Resolution Graph Neural Network

- Use the concatenation of the output after different layer of GNNs to keep multi-scale (multi-hop) substructure information.
- Each layer of graph neural network captures one-hop neighbour relationship.



Graph of Graphs Neural Network

Local graph neural network:

The neural network used to learn representations for chemical molecules. We use l -layer graph convolutional network (GCN) as the local graph neural network:

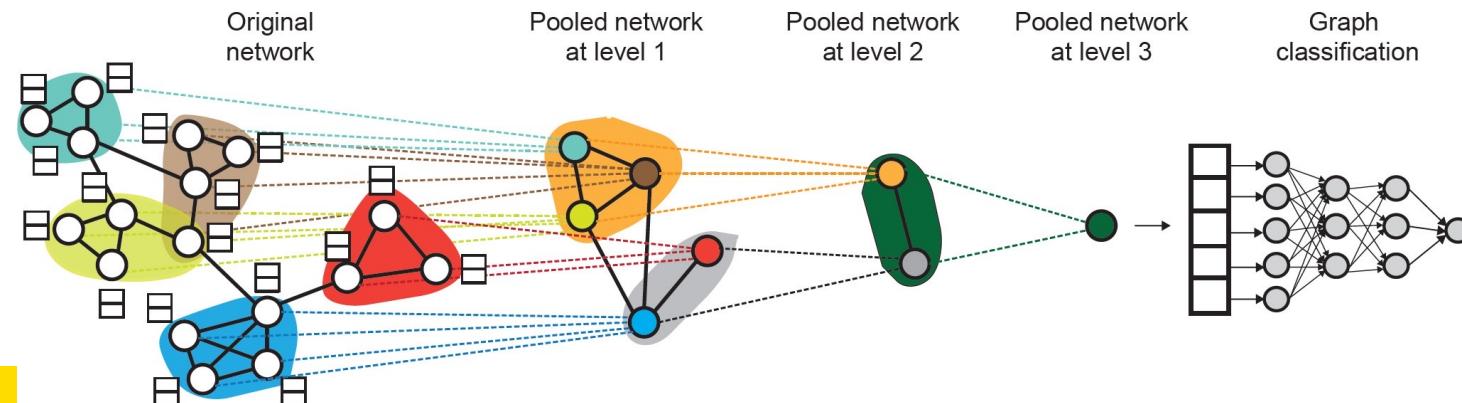
$$M_{(l+1)} = GCN_l(A, M_l)$$
$$GCN_l(A, M_l) = \sigma\left(D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}M_lW_l\right)$$



Graph of Graphs Neural Network

Graph pooling

Graph pooling is a computational strategy to reduce the number of graph nodes; in this way, one has a unified graph-level rather than node-level representation for graph-structured data while the size and topology of an individual graph are changing.



Graph of Graphs Neural Network

Graph pooling examples

- (1) Global mean pooling

$$y_0 = \text{Mean}(\{\mathbf{h}_v \in \mathbb{R}^d, \forall v \in G\})$$

- (2) Global max pooling

$$y_0 = \text{Max}(\{\mathbf{h}_v \in \mathbb{R}^d, \forall v \in G\})$$

- (3) Global sum pooling

$$y_0 = \text{Sum}(\{\mathbf{h}_v \in \mathbb{R}^d, \forall v \in G\})$$

- Issue: Global pooling over a (large) graph will lose information



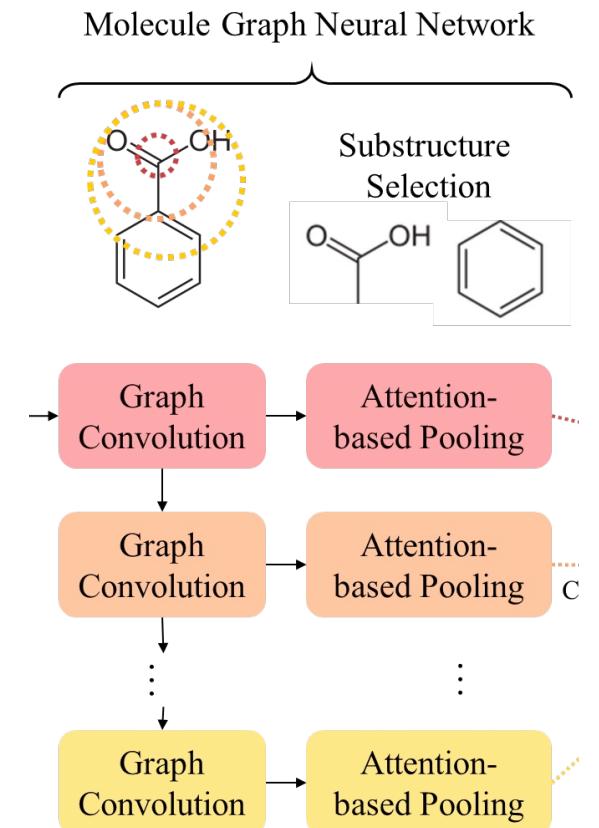
Graph of Graphs Neural Network

Graph attention pooling

- The attention-based pooling method to select **the most representative substructure** to represent the molecule graph:

$$s_l = \sigma \left(\widetilde{D}^{-\frac{1}{2}} \widetilde{\tilde{A}} D^{-\frac{1}{2}} M_l W_{att}^l \right)$$

$idx = top(s, [\gamma n]),$
 $s_{mask} = s_{idx}, M_{sel} = M \odot s_{mask}$

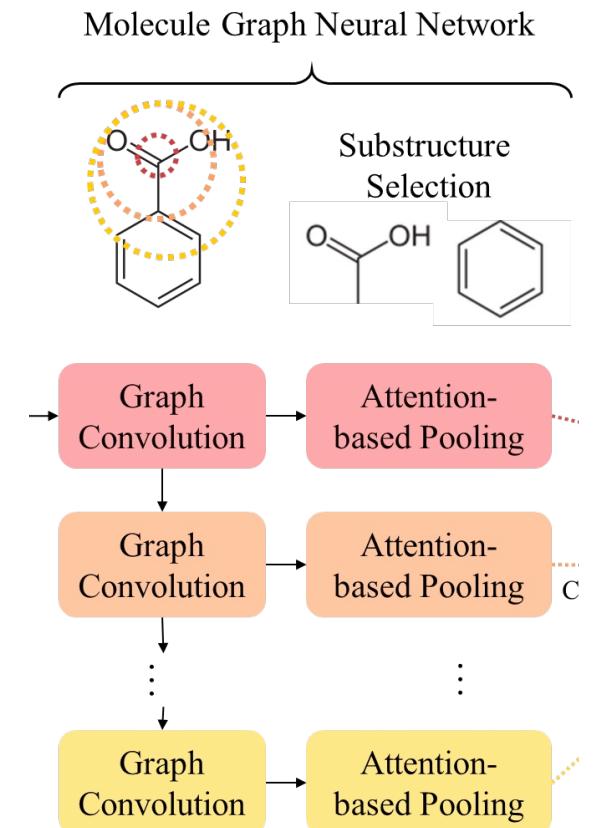


Graph of Graphs Neural Network

Graph attention pooling

$$s_l = \sigma\left(D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}M_lW_{att}^l\right)$$
$$idx = top(s, [\gamma n]),$$
$$s_{mask} = s_{idx}, M_{sel} = M \odot s_{mask}$$

- M_{sel} is the representation matrix for the selected atoms in the molecule graph
- After selection, the combination of mean pooling and sum pooling is used to produce the representation for the molecule graph, which is also the input for interaction graph neural network.

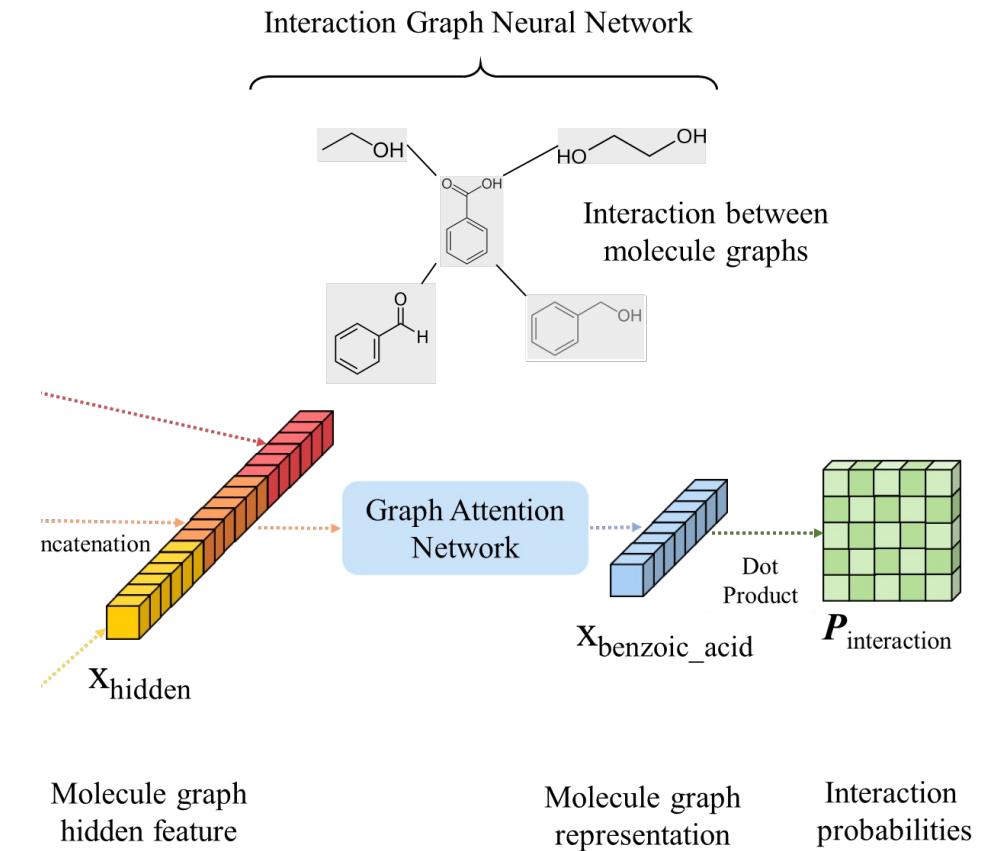


Graph of Graphs Neural Network

Graph attention network:

- With the learned molecule graph representations, the embedding is updated based on the interaction network. The multi-head attention mechanism is utilized:

$$x_{G_i}^{l+1} = \left\|_{\kappa=1}^K \sigma \left(\sum_{j \in \eta_{G_i}} \alpha_{ij}^\kappa W_\kappa^l x_{G_j}^l \right) \exp \left(\text{LeakyRelu} \left(a [Wx_{G_i}] || [Wx_{G_j}] \right) \right) \right\|_{\kappa=1}^K$$
$$\alpha_{ij} = \frac{\exp \left(\text{LeakyRelu} \left(a [Wx_{G_i}] || [Wx_{G_j}] \right) \right)}{\sum_{n \in \eta_{G_i}} \exp \left(\text{LeakyRelu} \left(a [Wx_{G_i}] || [Wx_{G_n}] \right) \right)}$$



Graph of Graphs Neural Network

Training objectives

The predicted interaction probability:

$$p_{ij} = \sigma(x_{G_i}^T \cdot x_{G_j})$$

Cross-entropy loss function:

$$\mathcal{L}_{CCI} = \sum_{(G_i, G_j) \in G_{CCI}} -\log(p_{ij}) - E_{m \sim P_j} \log(1 - p_{im})$$



Graph of Graphs Neural Network

Datasets:

- CCI tasks: The CCI dataset assigns a score from 0 to 999 to describe the interaction probability where a higher score indicates higher interaction probability. According to threshold score, we get two datasets with chemical interaction probability score over 900 and 950: **CCI900** and **CCI950**.
- DDI tasks. For the drug-drug interaction prediction problem, **DDI** dataset and the side effect dataset **SE** are used. A vector representation (attribute) is assigned to each side effect type produced by pre-trained BERT model



Graph of Graphs Neural Network

Experiment results:

	CCI900		CCI950	
	AUC	AP	AUC	AP
DeepCCI	0.925	0.918	0.957	0.957
DeepDDI	0.891	0.886	0.916	0.915
MR-GNN	0.927	0.921	0.934	0.924
MLRDA	0.922	0.907	0.959	0.948
SEAL	0.894	0.886	0.941	0.937
GoGNN	0.937	0.932	0.963	0.962
GoGNN-M	0.914	0.909	0.938	0.931
GoGNN-I	0.921	0.898	0.929	0.912
GoGNN-noPool	0.931	0.930	0.958	0.954
GoGNN-noAttn	0.909	0.905	0.956	0.948

Table 1: Result of chemical-chemical interaction prediction task.

	DDI		SE	
	AUC	AP	AUC	AP
DeepCCI	0.862	0.856	0.819	0.806
DeepDDI	0.915	0.912	0.827	0.809
MR-GNN	0.932	0.922	0.769*	0.752*
MLRDA	0.931	0.926	0.847*	0.825*
Decagon	-	-	0.872	0.832
SEAL	0.925	0.921	N/A	N/A
GoGNN	0.943	0.933	0.930	0.927
GoGNN-M	0.905	0.902	0.862	0.817
GoGNN-I	0.922	0.917	0.860	0.834
GoGNN-noPool	0.900	0.891	0.912	0.909
GoGNN-noAttn	0.925	0.921	0.897	0.883

* indicates that the result is the output of the baselines after two weeks' training.

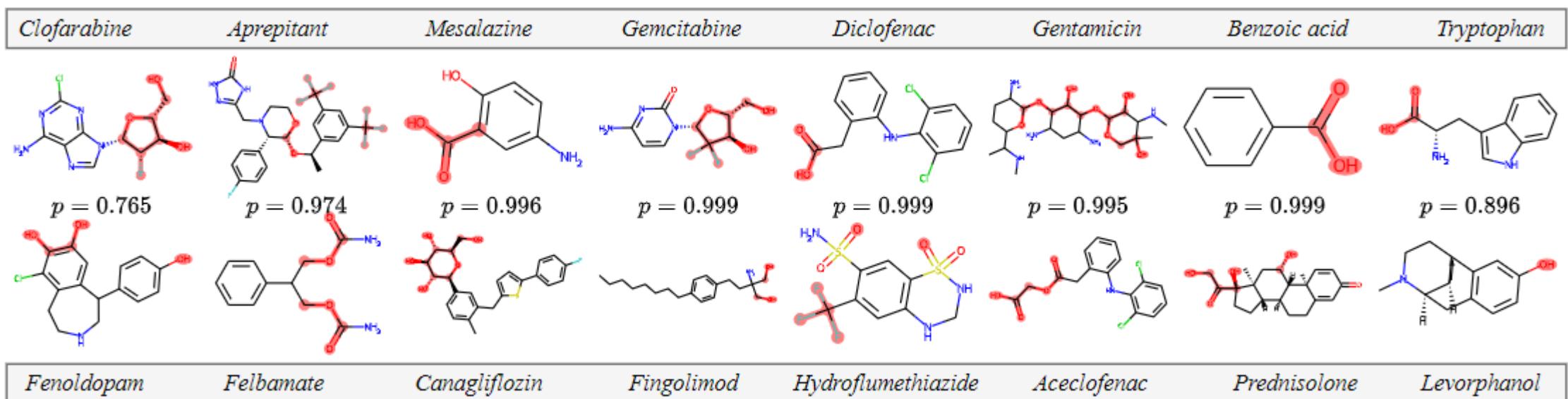
- DDI dataset has no protein data which is required by Decagon

Table 2: Result of drug-drug interaction prediction task.

Graph of Graphs Neural Network

Case study:

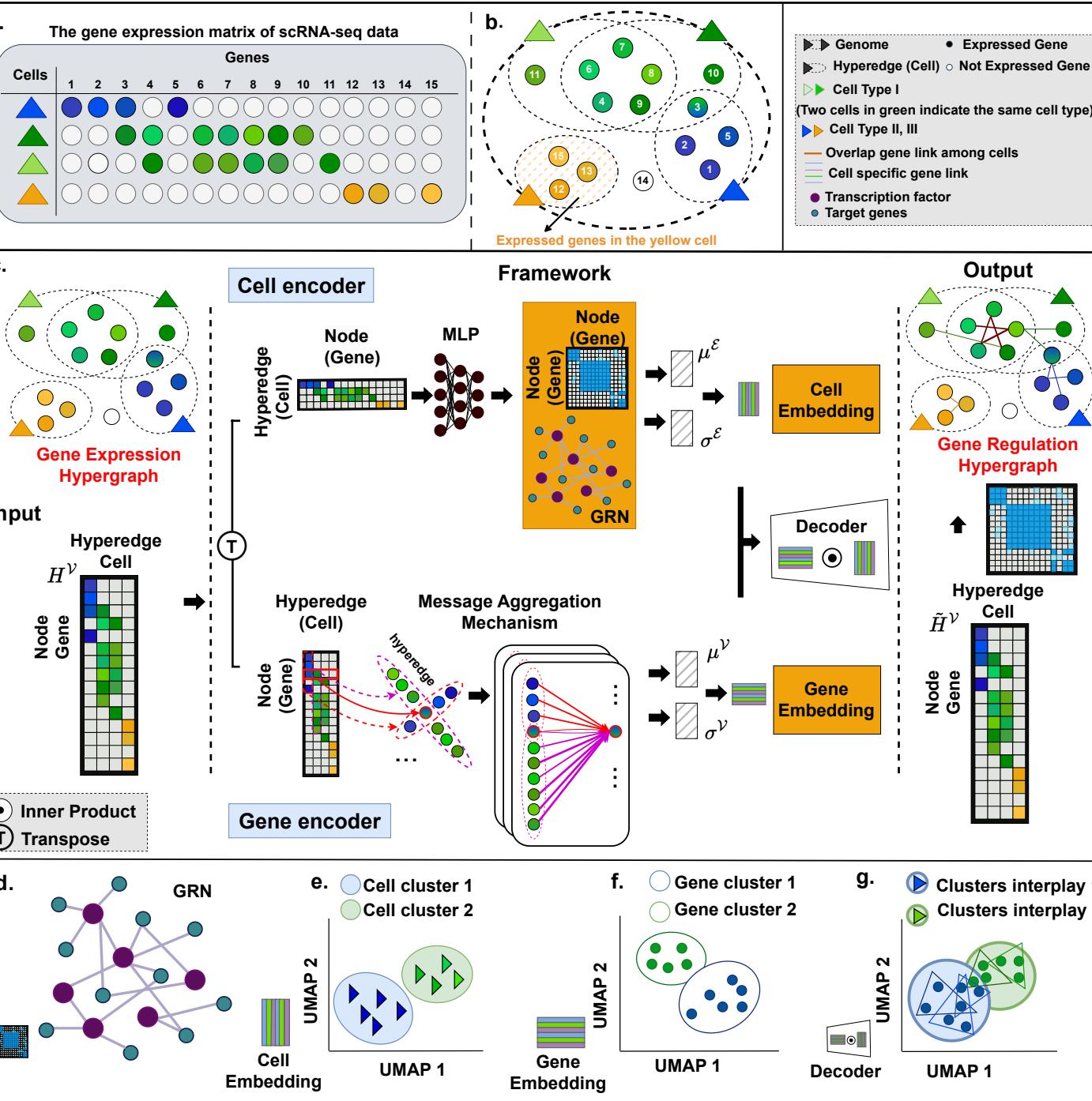
Red: the substructures that are responsible of the interactions.



Our recent work

- "Denoising Variational Graph of Graphs Auto-Encoder for Predicting Structured Entity Interactions" -- TKDE 2023

Gene Regulatory Network



Hypergraph Neural Network

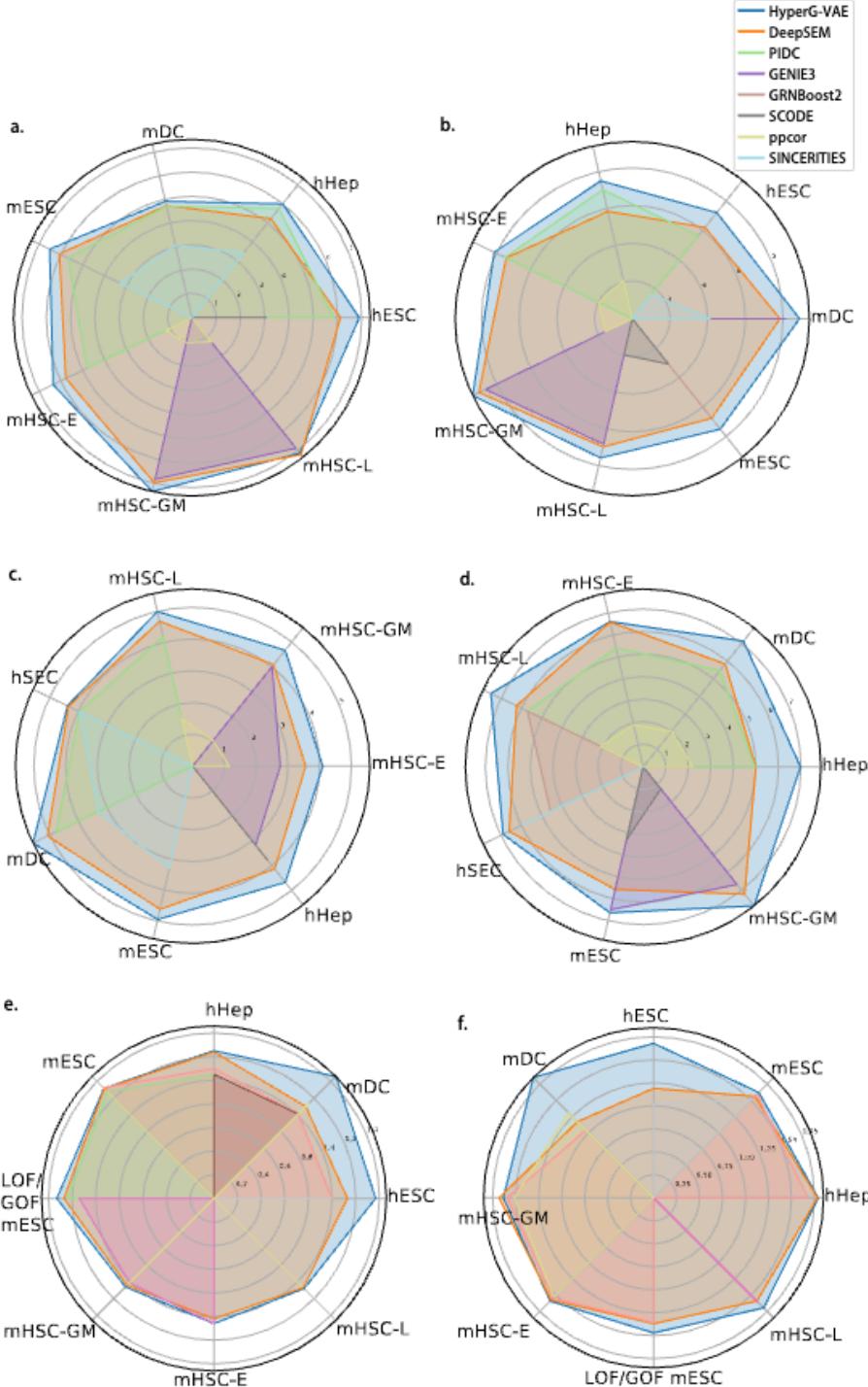


"Inferring gene regulatory networks by hypergraph generative model."

Gene Regulatory Network

Hypergraph Neural Network

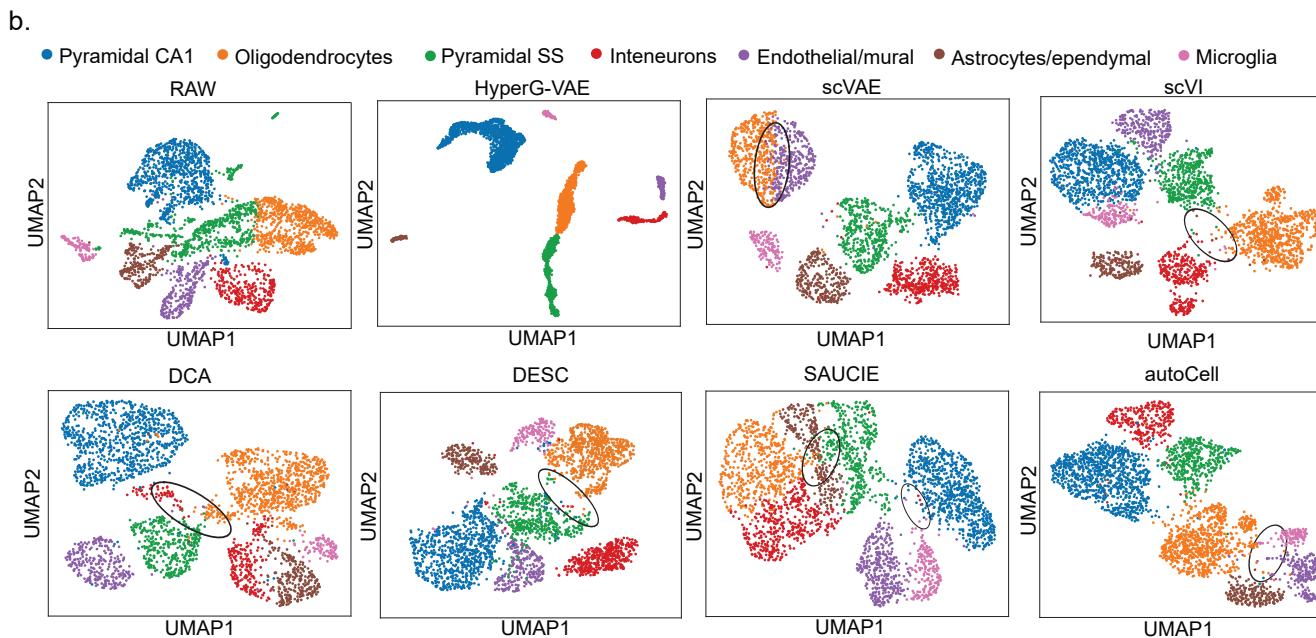
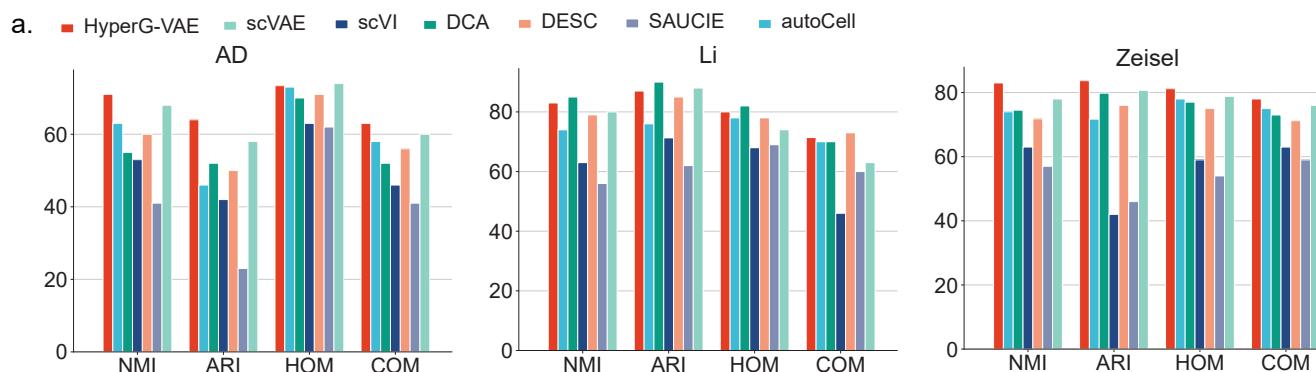
GRN inference



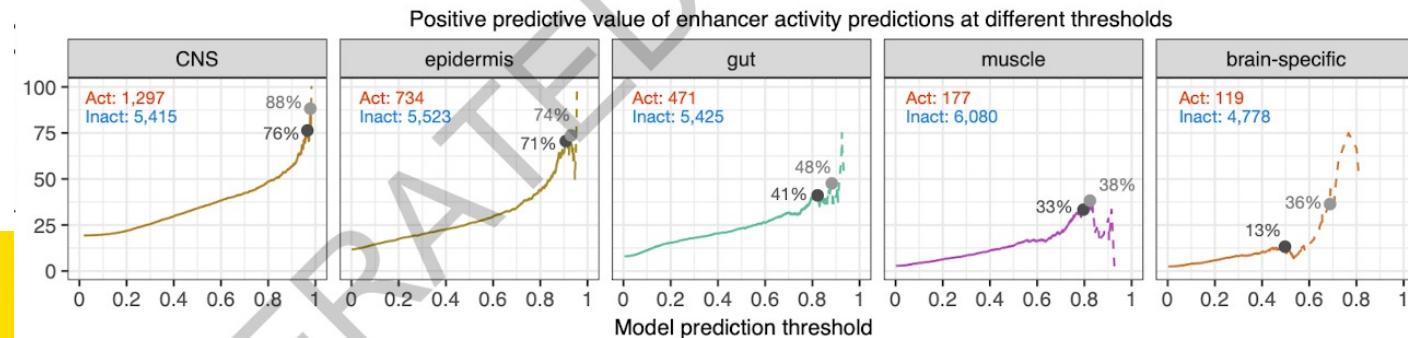
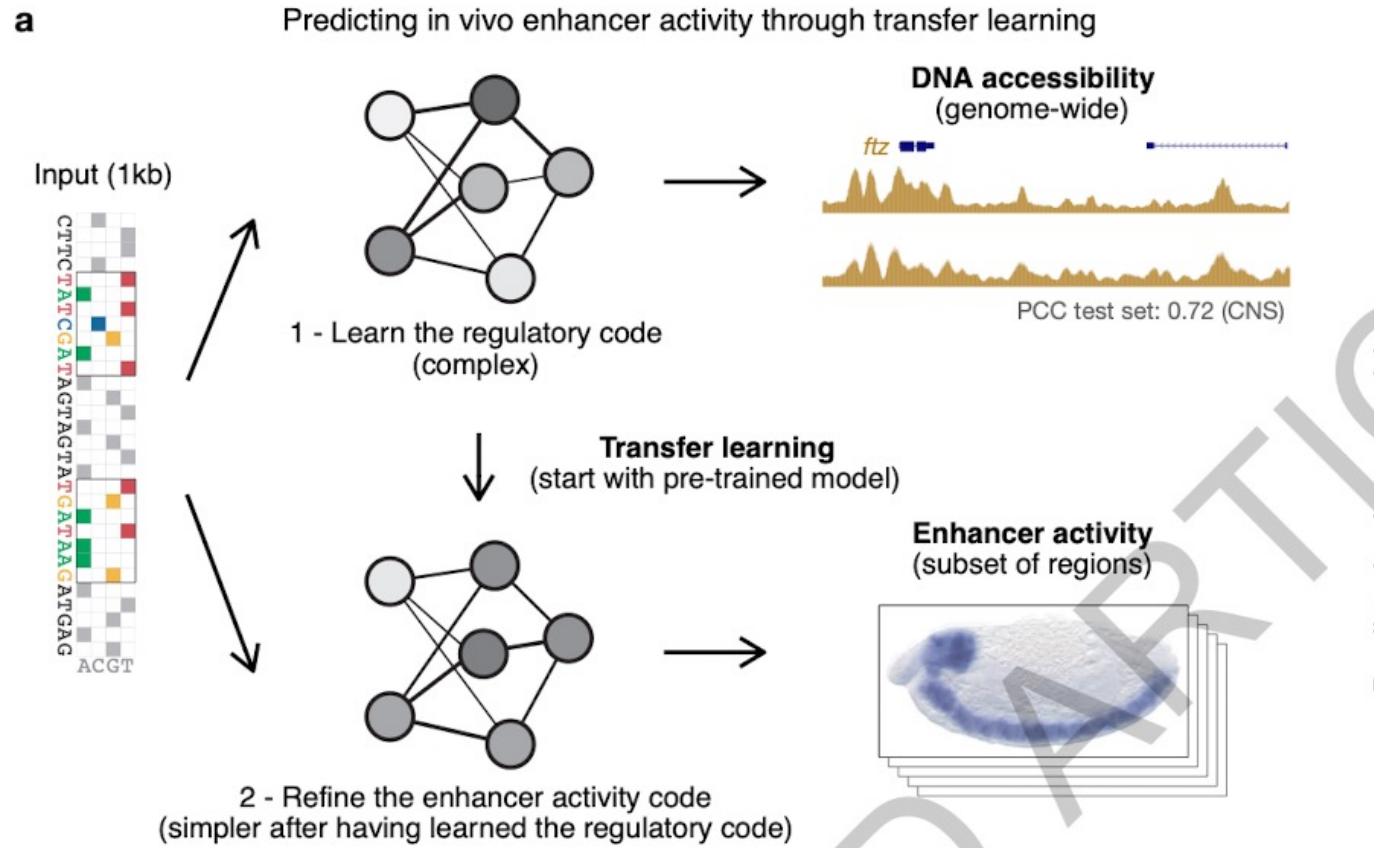
Gene Regulatory Network

Hypergraph Neural Network

Single Cell Clustering



Gene Expression Prediction

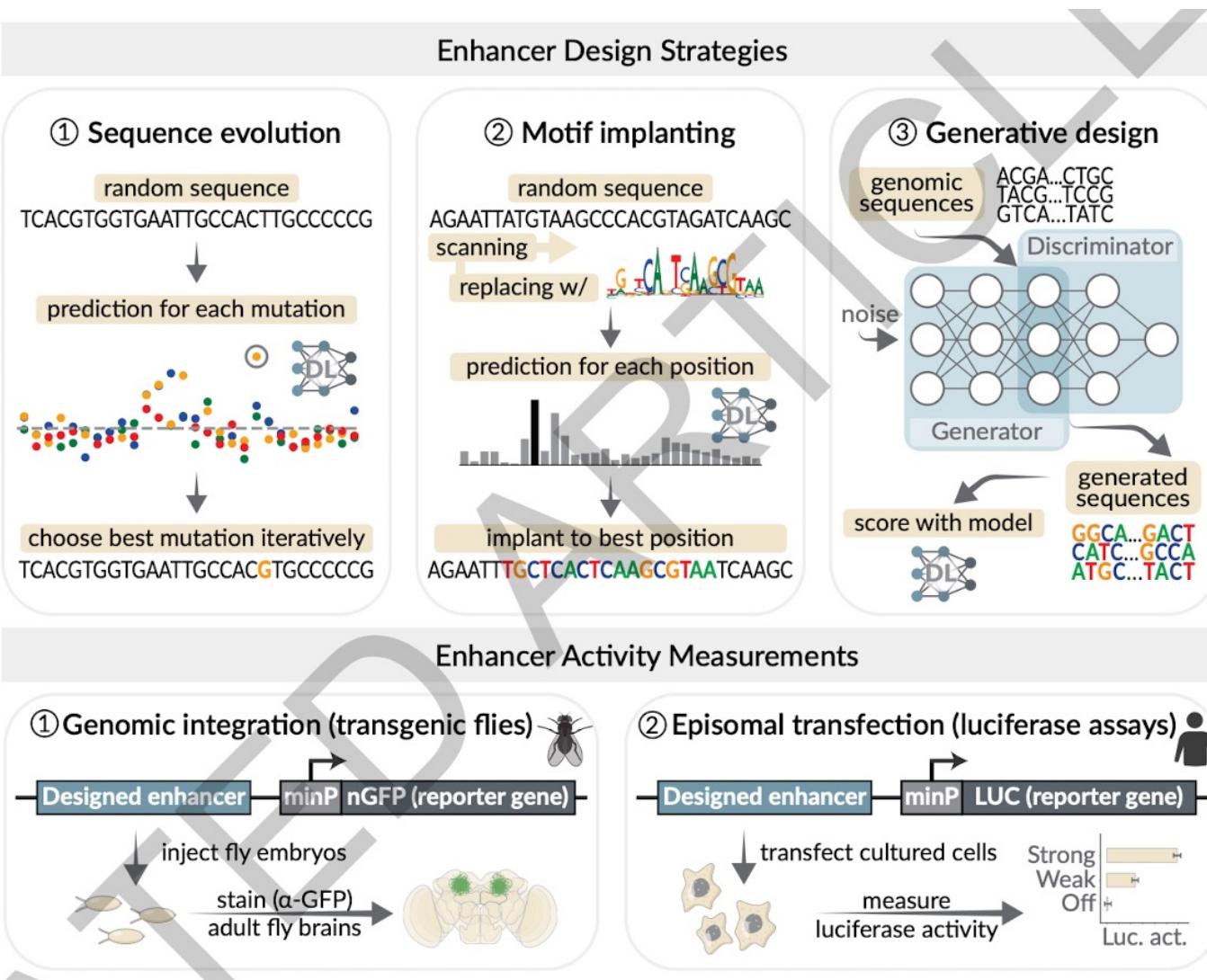


Related work

Cell-type-specific enhancer design



Gene Expression Prediction



Related work

Cell-type-specific enhancer design

Gene Expression Prediction

Normally, the single cell gene expression sequences are obtained in a control condition, e.g., the mouse are kept in the cage with the normal food and water.

However, the gene expression in the specific environment is crucial to find the biological pathway.

For example, how will the gene expression change when the mouse are in the enriched environment with sufficient food and entertaining toys.

How will the gene expression change when the mouse are shocked by electricity.



Cell specific gene expression prediction in different conditions



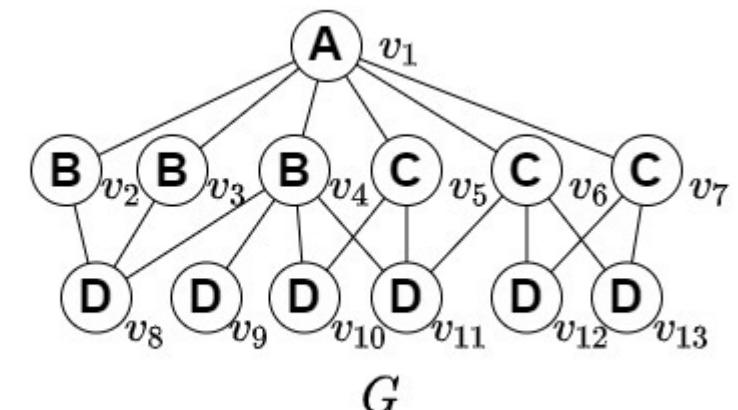
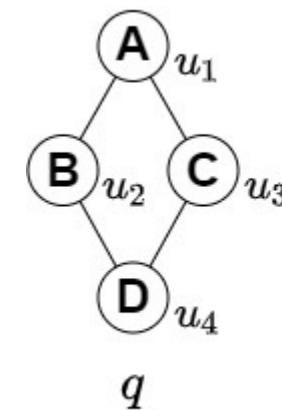
Subgraph Isomorphism



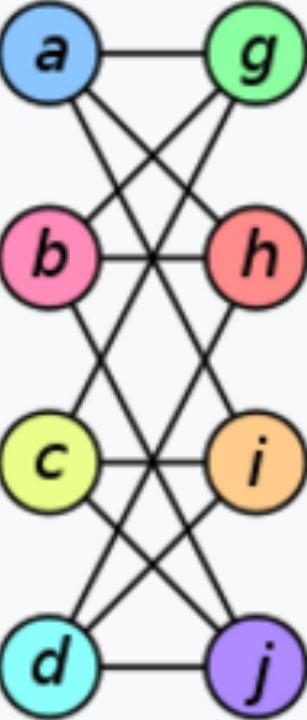
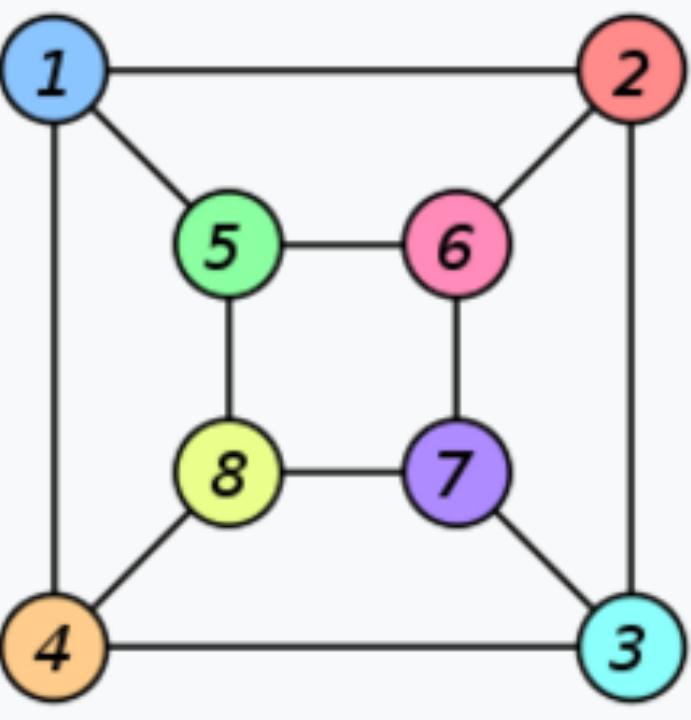
Subgraph Counting

Subgraph Isomorphism

- Query graph $q = (V, E, f_l)$
- Data graph $G = (V', E', f_l)$
- Subgraph Isomorphism: injective function $f_{iso}: V \rightarrow V'$:
 - $\forall u \in V, f_l(u) = f_l(f_{iso}(u))$
 - $\forall e(u, u') \in E, e(f_{iso}(u), f_{iso}(u')) \in E'$
- Determining the existence of subgraph isomorphism is **NP-complete**.

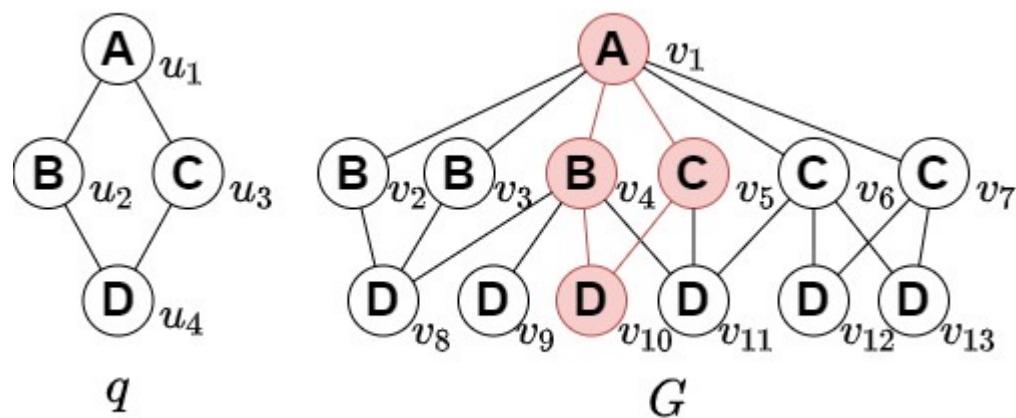


Graph isomorphism (a more complicated example)

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Subgraph Counting

Subgraph Counting: Given a query graph q and a data graph G , the problem is to count the number of subgraphs in the data graph that match the query graph by subgraph isomorphism.

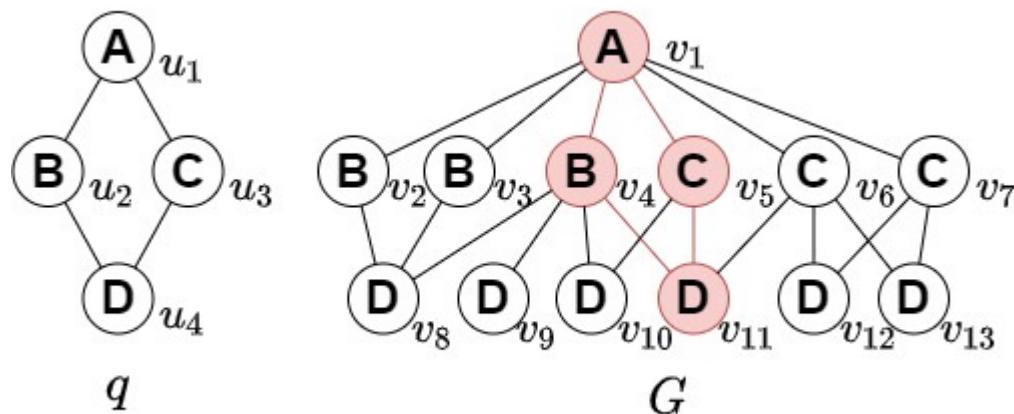


Subgraph isomorphisms

1. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{10})$

Subgraph Counting

Subgraph Counting: Given a query graph q and a data graph G , the problem is to count the number of subgraphs in the data graph that match the query graph by subgraph isomorphism.

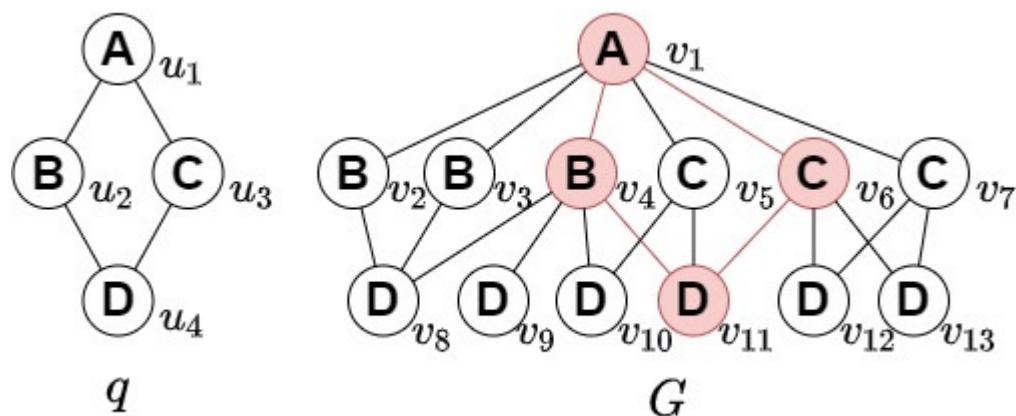


Subgraph isomorphisms

1. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{10})$
2. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{11})$

Subgraph Counting

Subgraph Counting: Given a query graph q and a data graph G , the problem is to count the number of subgraphs in the data graph that match the query graph by subgraph isomorphism.



Subgraph isomorphisms

1. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{10})$
2. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{11})$
3. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_6, v_{11})$

Why Subgraph Counting?

Applications

Analysis on Social Networks:

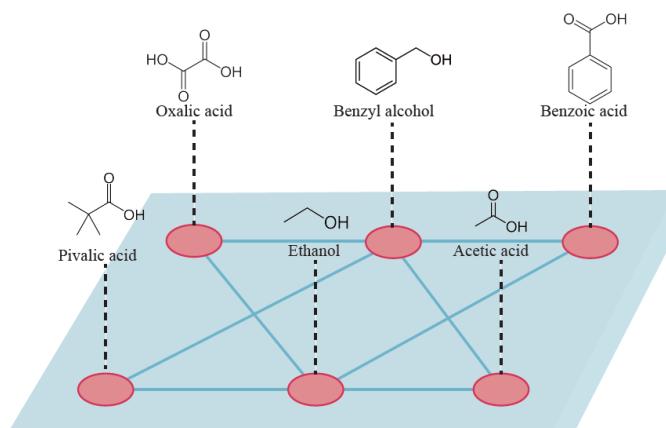
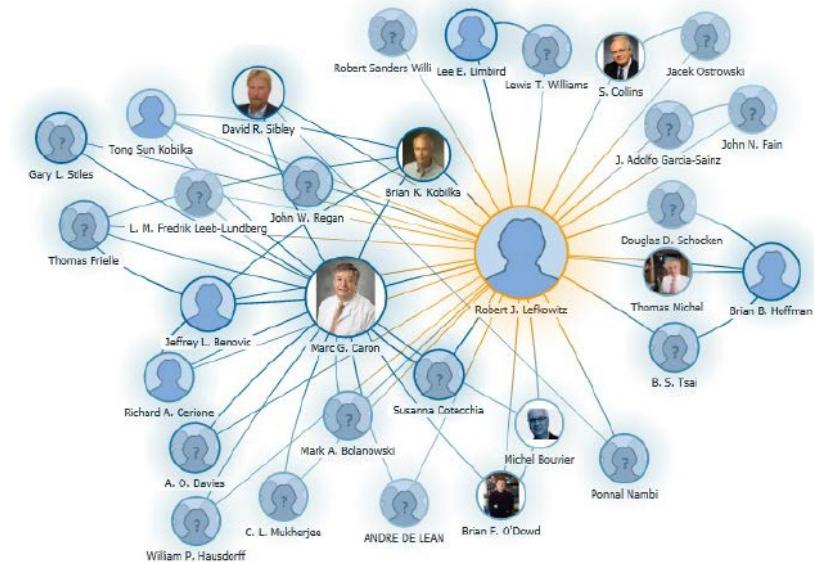
- Find co-authorships, community detection.

Analysis on Biological Networks:

- On brain, regulation, protein and molecule graphs.
- Summarize the structural patterns for the biological graphs.

Query Optimization for Subgraph Matching Queries:

- Cardinality estimation for multi-way join.



Existing Subgraph Counting Methods

Algorithmic Methods:

Enumeration-based methods:

- Computational complexity.

Sampling-based methods:

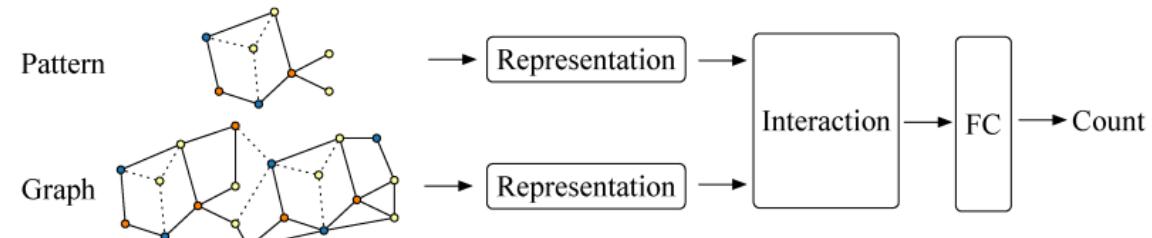
- Sampling failure.

Summary-based methods:

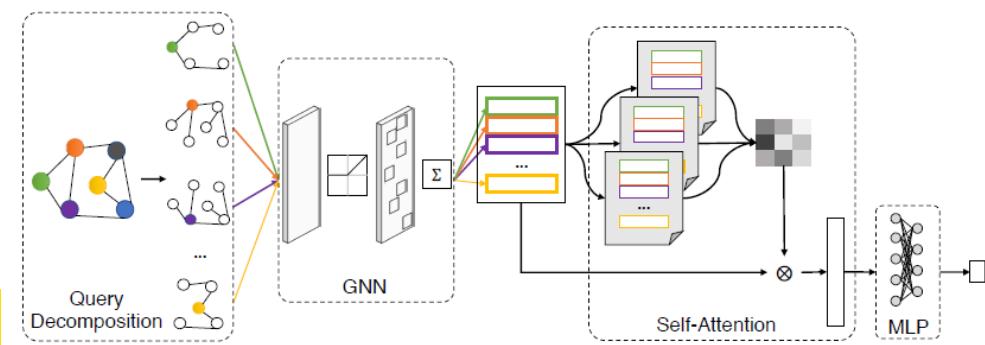
- Independent assumption.

Learning-based Methods:

Neural Subgraph Isomorphism Counting



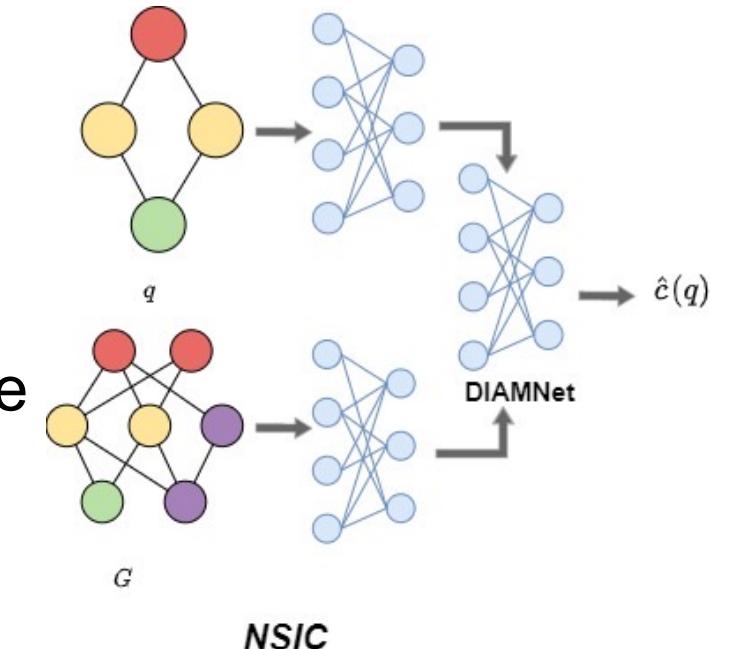
A Learned Sketch for Subgraph Counting



Existing Subgraph Counting Methods

Neural Subgraph Isomorphism Counting:

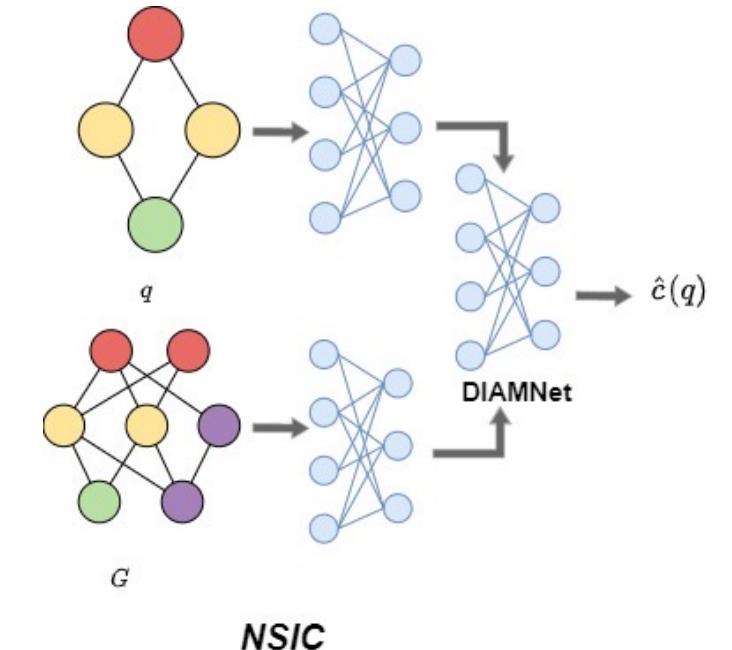
- The query graph and data graph are input into the graph neural networks for representation learning.
- The learned representations are input into the RNN-based network named DIAMNet to predict the subgraph counts.



Existing Subgraph Counting Methods

Neural Subgraph Isomorphism Counting:

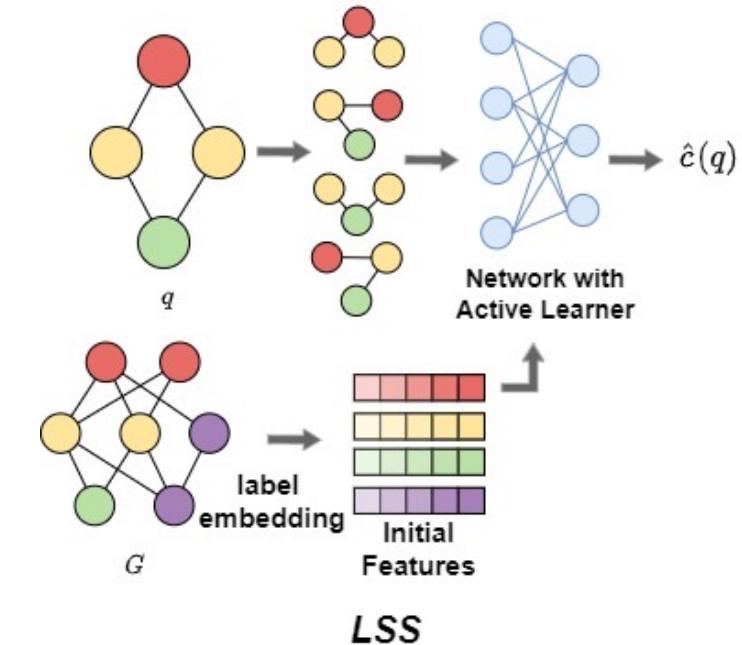
- The data graph is usually large-scale.
- The model will face the efficiency and scalability issue.
- Since the data graph contains more information, it is hard to distinguish the counting results of different query graphs when the data graph is large. The representation of data graph will dominate the computation in this case.



Existing Subgraph Counting Methods

A Learned Sketch for Subgraph Counting:

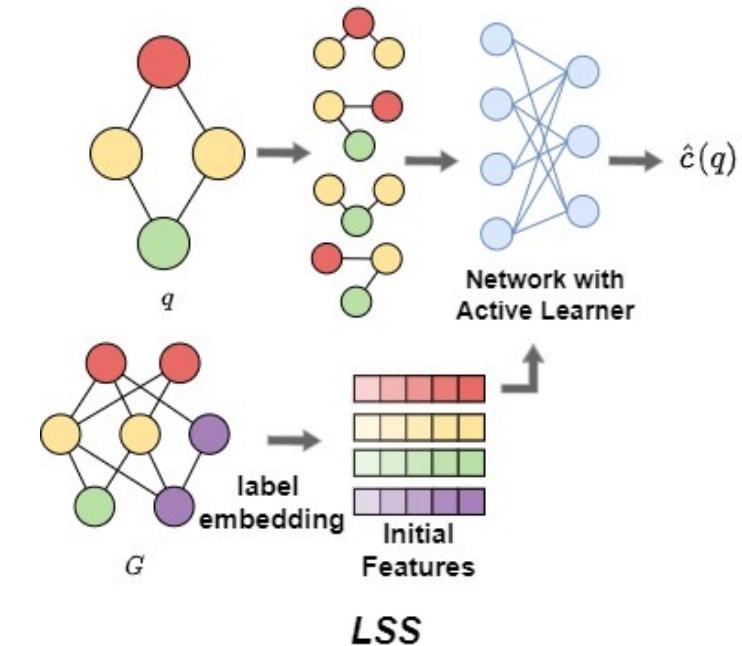
- The initial features of query nodes are computed based on the labels and structure of data graph by the graph embedding methods like DeepWalk.
- The query graph is decomposed into small substructures and fed into the graph neural network with active learner to predict the subgraph counts.



Existing Subgraph Counting Methods

A Learned Sketch for Subgraph Counting:

- Cannot fully utilize the data graph information. The topological information of data graph is somehow ignored.
- Consequently, the model has limited robustness, i.e., the result can be easily affected by minor modification on the query graph.

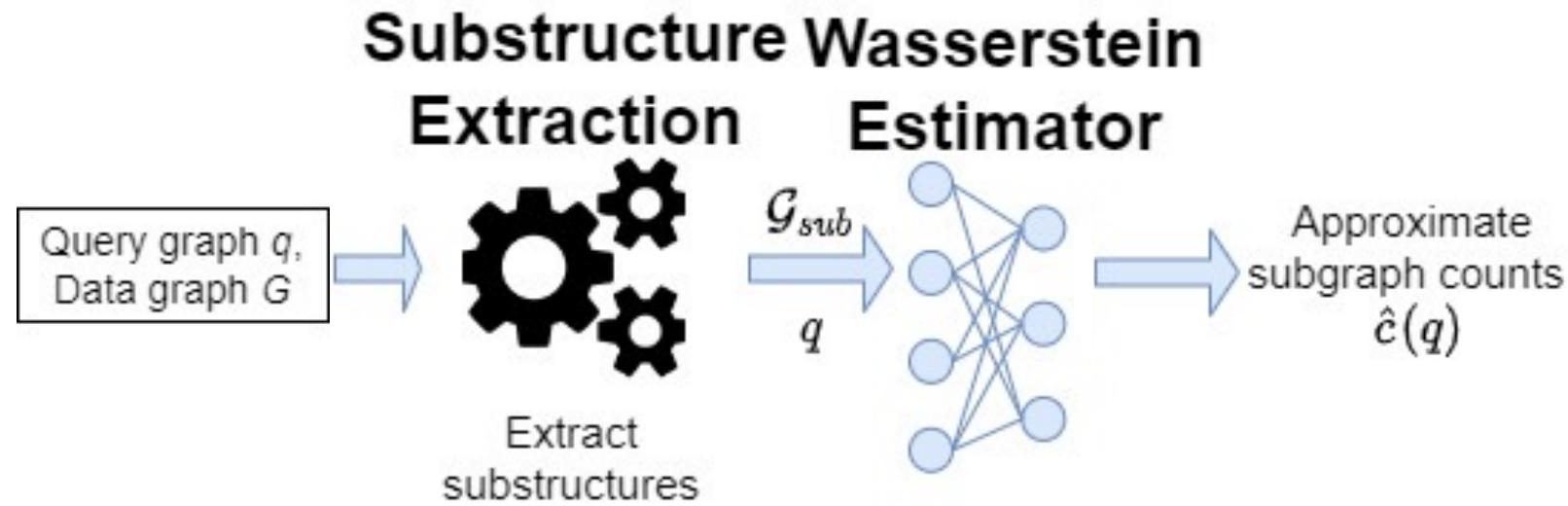


Goal

- Utilize both query and data graph information.
- Avoid efficiency and scalability issue.
- Improve the subgraph counting accuracy.



Neural Subgraph Counting method: NeurSC



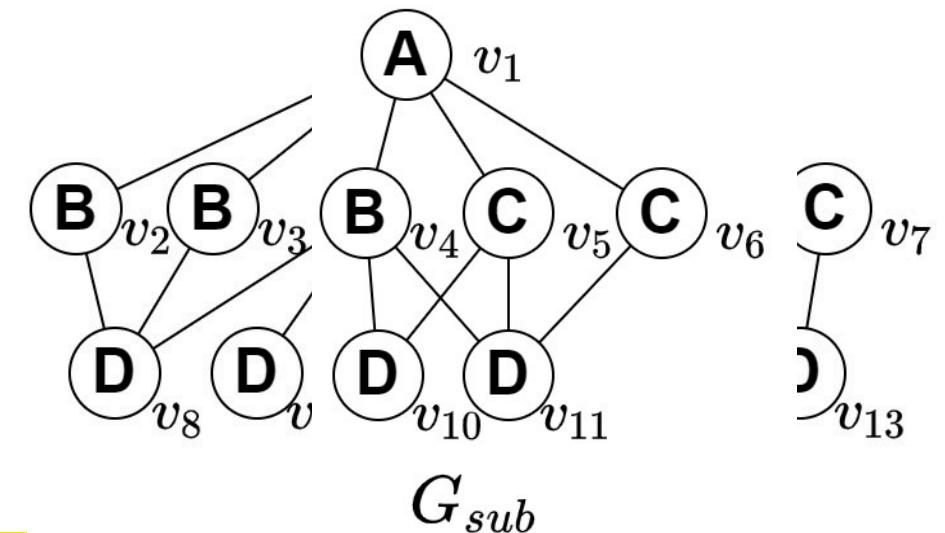
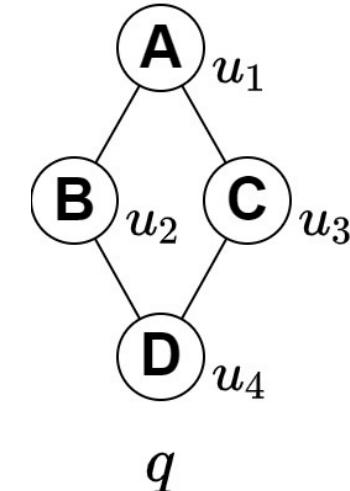
Our recent work

- "Neural Subgraph Counting with Wasserstein Estimator" -- SIGMOD 2022

NeurSC

Substructure Extraction

- **Complete Candidate Vertex Set (CS):**
 - $CS(u)$ for query vertex $u \in V$ is a set of data vertices $v \in V'$
 - If (u, v) exists in a match from q to G , then $v \in CS(u)$
- First, we determine the complete candidate vertex set for all query vertices using *local pruning* and *global refinement*.
- Based on **neighboring** and **label** information
- Candidate set of query q : $CS(q) = \cup_{u \in V} CS(u)$
- Induced subgraph of G with vertices $CS(q)$ is used as the candidate substructures, denoted as G_{sub}



NeurSC

Feature Initialization

$$\mathbf{x}_v = f_b(\deg_v) \parallel f_b(f_l(v)) \parallel \underset{i=1}{\overset{k}{MP}}_{\forall v' \in N^{(i)}(v)} f_b(\deg_{v'}) \parallel f_b(f_l(v'))$$

Degree information Label information Neighbor information

- \parallel denotes the concatenation.
- f_b denotes the binary encoding that converts the decimal digits into binary numbers.
- MP denotes the mean pooling.
- $N^{(i)}(v)$ denotes the i -hop neighbors of v .



NeurSC

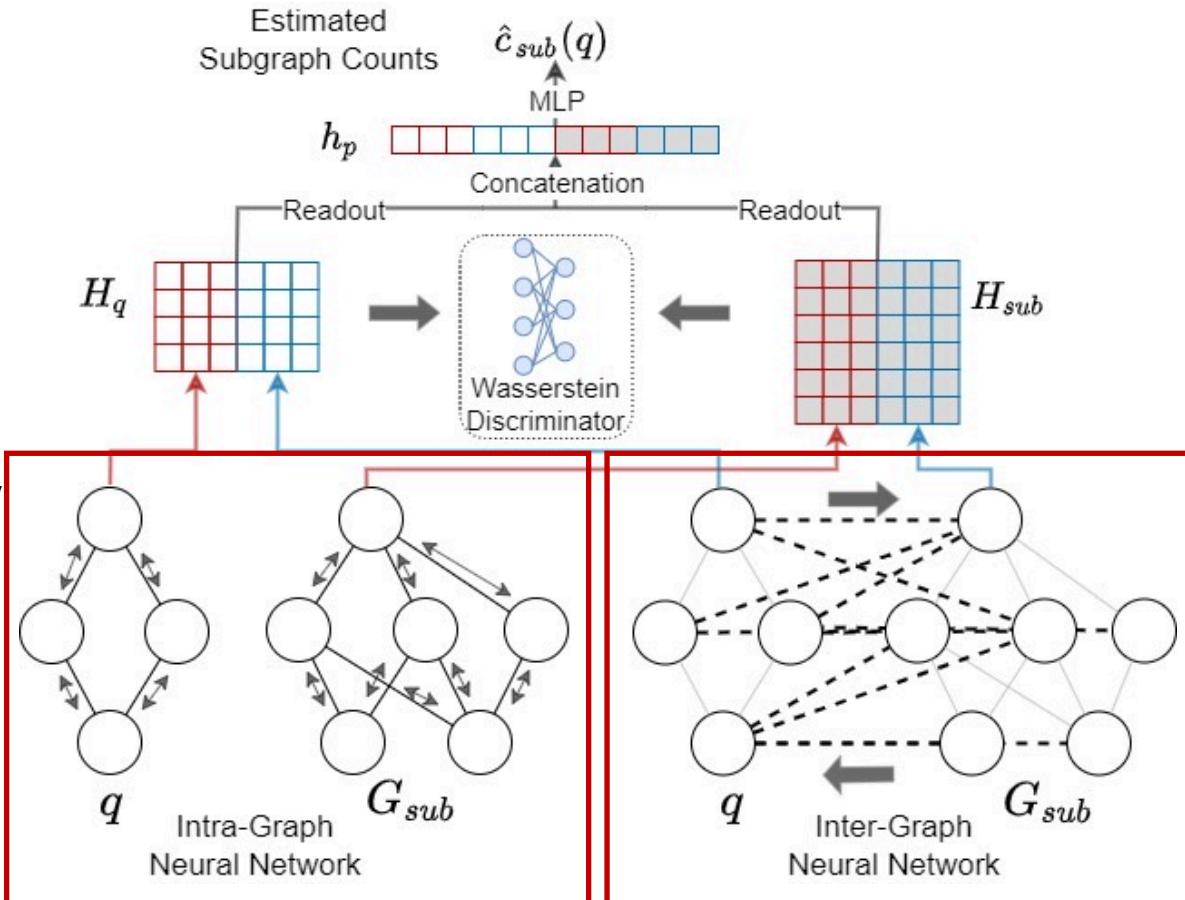
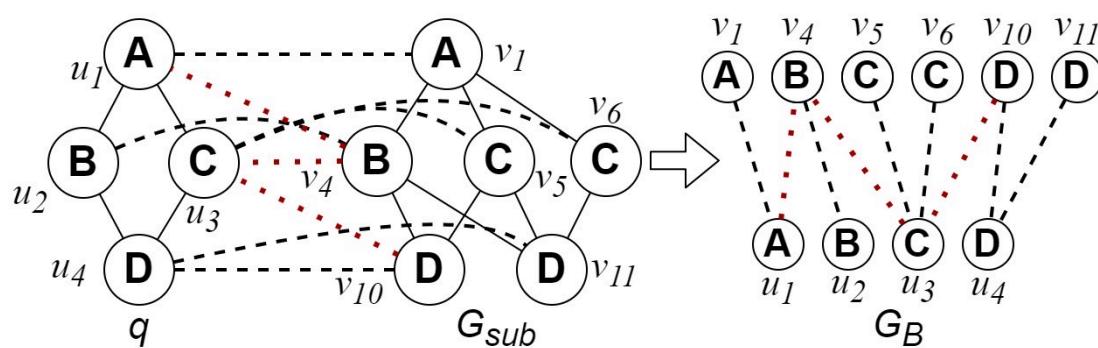
Wasserstein Estimator

- Intra-Graph Neural Network**

- For both query graph and substructure.
- Capture structural and attribute information.
- $\mathbf{h}_u^{(k)} = \text{MLP}^{(k)}((1 + \epsilon^{(k)})\mathbf{h}_u^{(k-1)}, \sum_{u' \in N_q(u)} \mathbf{h}_{u'}^{(k)})$

- Inter-Graph Neural Network**

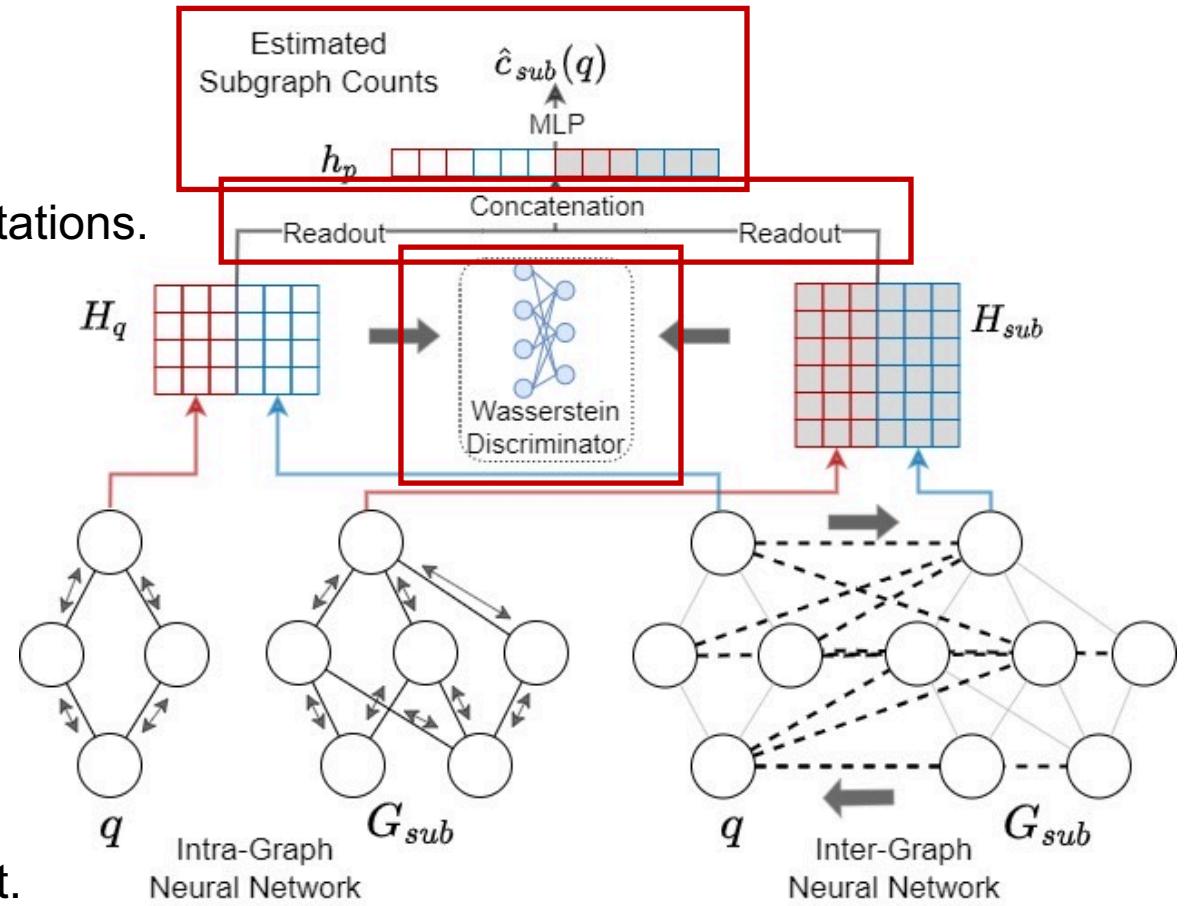
- Construct a bipartite graph for inter-relationship.
- Capture the mapping relationship between query vertices and corresponding candidate vertices
- $\mathbf{h}_u^{(k)} = \sigma(a_{uu}^{(k)} \theta^{(k)} \mathbf{h}_u^{(k-1)}, \sum_{v \in N_{G_B}(u)} a_{uv}^{(k)} \theta^{(k)} \mathbf{h}_v^{(k)})$



NeurSC

Wasserstein Estimator

- **Readout**
 - Sum Pooling
 - Concatenation of intra- and inter-graph representations.
- **Prediction**
 - Multi-layer perceptron.
- **Wasserstein Discriminator**
 - Minimize Wasserstein distance between q and G
 - Further utilize the vertex correspondence information between q and G
 - $L_w(q, G_{sub}) = \sum_{u \in V'(q)} f_{\omega(h_u)} - \sum_{v \in V'(G_{sub})} f_{\omega(h_v)}$
- **Expressive Power**
 - *WEst* is as powerful as 1-Weisfeiler-Lehman test.



NeurSC

Learning objective and training procedure

- **q-error loss**

- $L_c(q) = \max\left(\frac{c(q)}{\hat{c}(q)+\varepsilon}, \frac{\hat{c}(q)}{c(q)}\right)$

- Optimize the model by reducing the prediction error.

- **Wasserstein loss**

- $L_w(q, G_{sub}) = \sum_{u \in V'(q)} f_\omega(h_u) - \sum_{v \in V'(G_{sub})} f_\omega(h_v)$

- Minimize the Wasserstein distance.

- **Overall loss**

- $L(q) = (1 - \beta)L_c(q) - \frac{\beta}{|G_{sub}|} \sum_{g \in G_{sub}} L_w(q, g)$

Algorithm 3: Training Procedure of *WEst*

Input: training query graph set Q_t , data graph G , estimation network f_θ , discriminator f_ω , learning rates $\alpha_\theta, \alpha_\omega$, batch size n_{batch} , number of training iterations $iter_\omega$.

```

1 Initialize optimizers  $opt_\theta, opt_\omega$  with learning rates  $\alpha_\theta, \alpha_\omega$ .
2 Separate  $Q_t$  into batches  $\{Q_b = \{q^{(i)}\}\}$  with  $n_{batch}$  query graphs.
3 for  $Q_b \in Q_t$  do
4   for  $q^{(i)} \in Q_b$  do
5     Generate  $\mathcal{G}_{sub}^{(i)}$  for  $q^{(i)}$ 
6      $X_q^{(i)} \leftarrow$  initial features of vertices in query graph  $q^{(i)}$ 
7     for  $j = 1, \dots, |\mathcal{G}_{sub}^{(i)}|$  do
8        $X_{sub}^{(j)} \leftarrow$  initial features of vertices in  $G_{sub}^{(j)}$ 
9        $H_q^{(i)}, H_{sub}^{(j)}, \hat{c}_j(q^{(i)}) \leftarrow f_\theta(q^{(i)}, G_{sub}^{(j)}, X_q^{(i)}, X_{sub}^{(j)})$ 
10      for  $iter_\omega$  do
11        Sample  $V'(q^{(i)})$  and  $V'(G_{sub}^{(j)})$ 
12        Update  $\omega$  by  $opt_\omega$  minimizing  $-\mathcal{L}_w$  in Eq. 9
13        Compute  $\mathcal{L}_w(q^{(i)}, G_{sub}^{(j)})$ 
14         $\hat{c}(q^{(i)}) = \sum_{j=1}^{|\mathcal{G}_{sub}^{(i)}|} \hat{c}_j(q^{(i)})$ 
15        Compute  $\mathcal{L}_c(q^{(i)})$  using Eq. 10
16        Update  $\theta$  by  $opt_\theta$  with  $\sum_{q^{(i)} \in Q_b} \mathcal{L}(q^{(i)})$  defined in Eq. 11

```

Experiment

Experimental Setup

- **Dataset**
 - 7 *data graphs* + 5 *query sets*.
- **Compared methods**
 - 5 non-learning methods.
 - 2 learning-based methods with 2 variants.
 - 2 variants of *NeurSC* for ablation study.
- **Parameter settings**
 - Initial dimension: 64
 - Hidden and output dimension 128
 - Intra- and Inter-GNN have 2 layers.
 - Prediction network is a 4-layer MLP.
- **Implementation**
 - Substructure extraction: C++
 - *WEst*: Python + Pytorch Geometric

Table 2: Statistics of Data Graphs

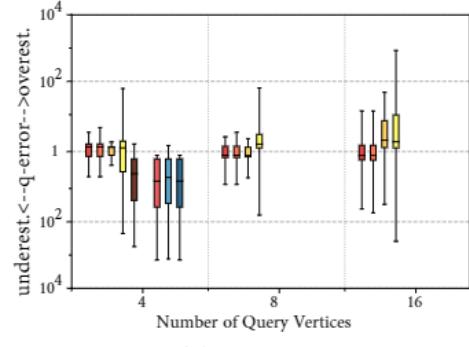
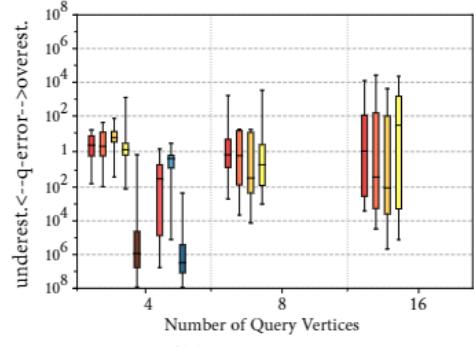
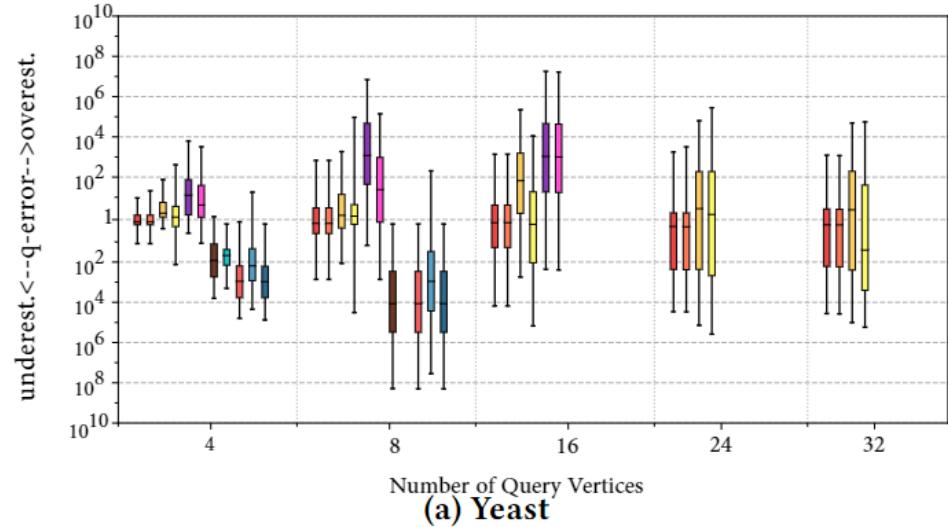
Dataset	V	E	L	d
Yeast	3,112	12,519	71	8.0
Human	4,674	86,282	44	36.9
HPRD	9,460	34,998	307	7.4
Wordnet	76,853	120,399	5	3.1
DBLP	317,080	1,049,866	15	6.6
EU2005	862,664	16,138,468	40	37.4
Youtube	1,134,890	2,987,624	25	5.3

Table 3: Details of Query Graphs

Dataset	# Queries	Query Sizes	Counts Range
Yeast	1,632	{4, 8, 16, 24, 32}	[10^0 , 10^{11}]
Human	339	{4, 8, 16}	[10^0 , 10^{10}]
HPRD	1,000	{4, 8, 16}	[10^0 , 10^4]
Wordnet	600	{4, 8}	[10^1 , 10^9]
DBLP	600	{4, 8}	[10^3 , 10^8]
EU2005	372	{4, 8}	[10^4 , 10^9]
Youtube	811	{4, 8, 16}	[10^0 , 10^{11}]

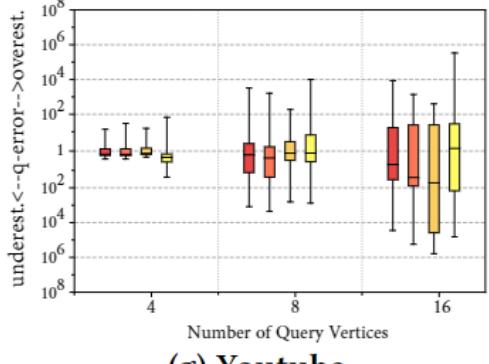
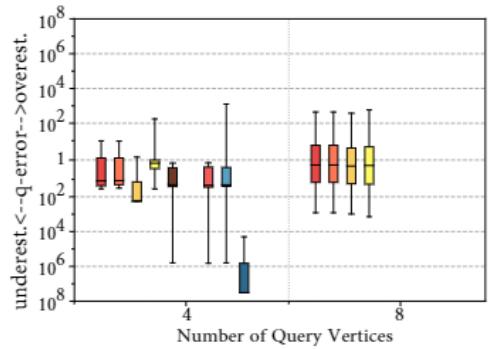
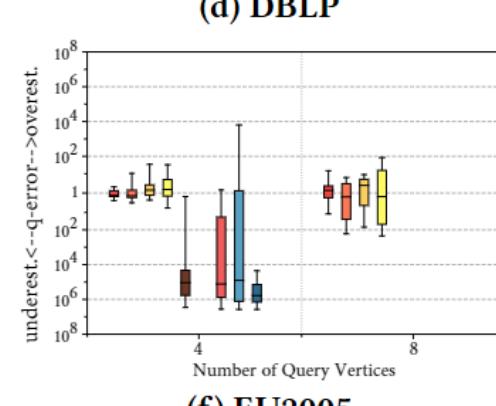
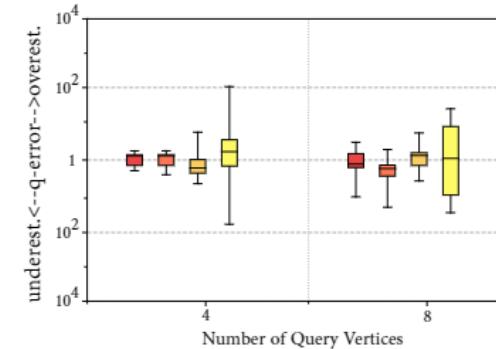
Experiment Results

Accuracy Evaluation



- Evaluation metric

- $q\text{-error}: \max\left(\frac{\max(1,c)}{\max(1,\hat{c})}, \frac{\max(1,\hat{c})}{\max(1,c)}\right)$



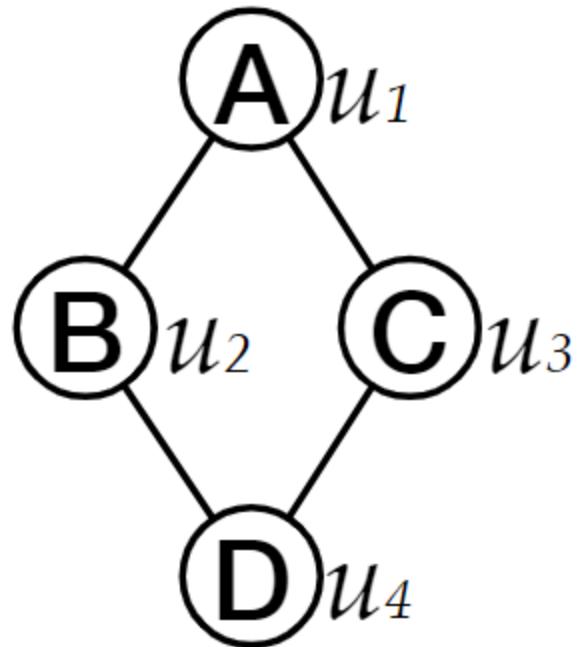
Subgraph Matching

The objective of the *subgraph matching* is searching for all *subgraph isomorphisms* from query graph q to data graph G

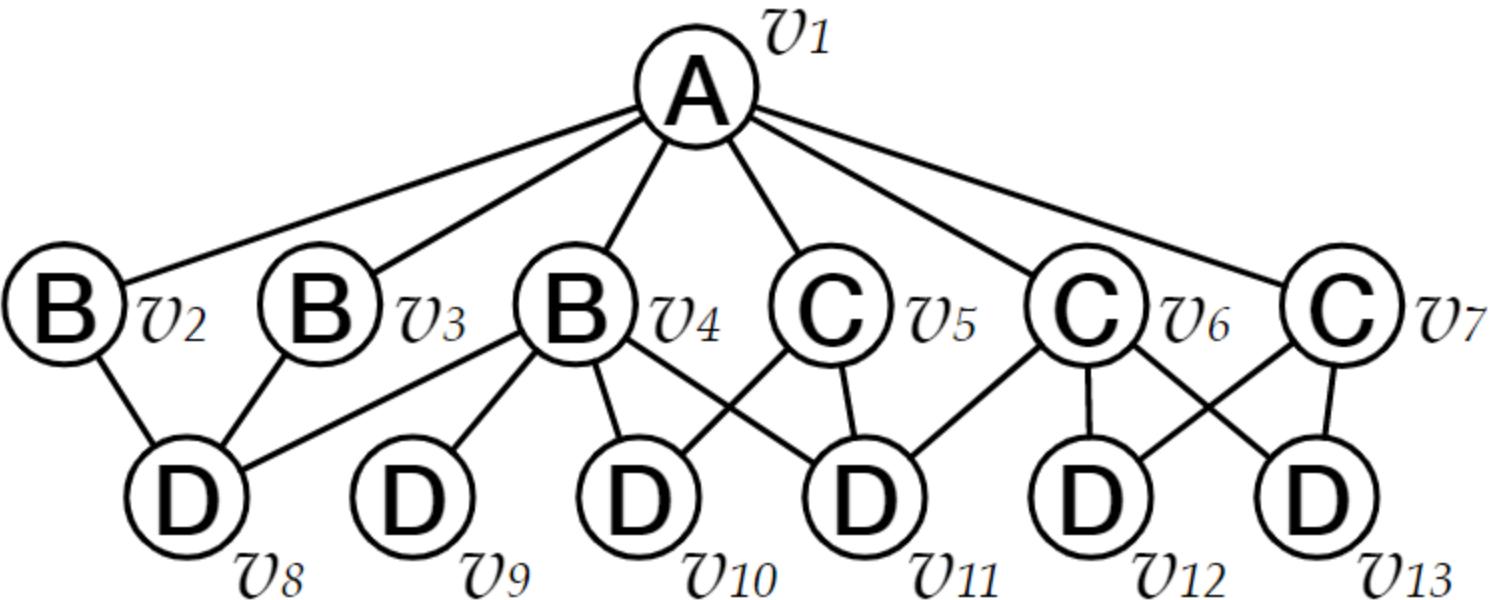
Definition II.1 (Subgraph Isomorphism). Given a query graph $q = (V, E)$ and a data graph $G = (V', E')$, a subgraph isomorphism is an injective function f_{iso} from V to V' such that (1) $\forall v \in V, f_l(v) = f_l(f_{iso}(v))$; and (2) $\forall e_{(u,v)} \in E, e_{(f_{iso}(u), f_{iso}(v))} \in E'$.



Subgraph Isomorphism



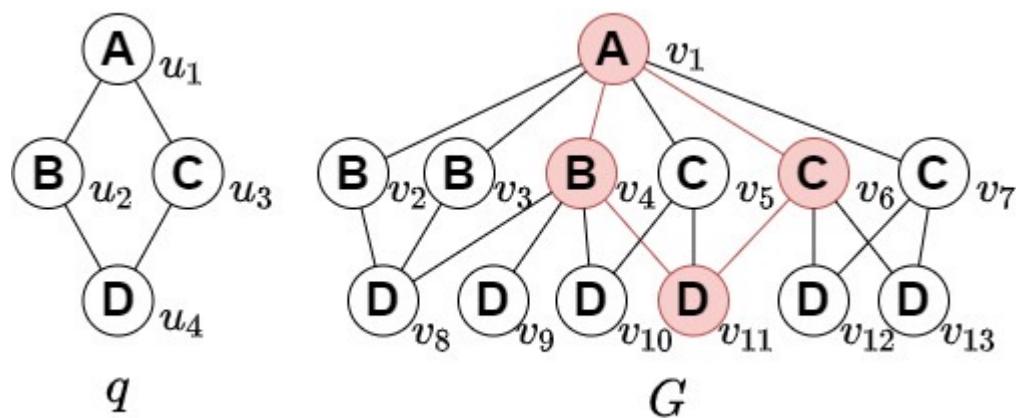
(a) Query Graph q



(b) Data Graph G



Subgraph Isomorphism



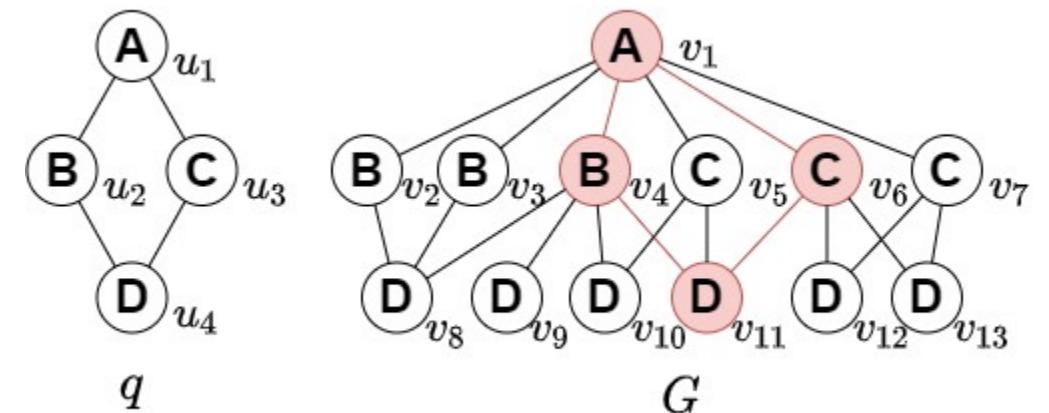
Subgraph isomorphisms

1. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{10})$
2. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{11})$
3. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_6, v_{11})$

Existing Subgraph Matching Methods

There are two major categories of subgraph matching methods:

- Backtracking-based methods
- Join-based methods



In this work, we focus on the backtracking-based methods.



Existing Subgraph Matching Methods

The backtracking-based methods can be partitioned in three main phases:

1. The complete candidate vertex set generation.
2. Matching order generation.
3. Matching enumeration.



Backtracking-based Methods

Complete candidate vertex set generation is to filter out the unpromising vertices, and hence reduce the search space before the enumeration process begins.

Definition II.2 (Complete Candidate Vertex Set \mathcal{C}). Given q and G , a complete candidate vertex set $C(u)$ of $u \in V(q)$ is a set of data vertices such that for each $v \in V(G)$, if (u, v) exists in a match from q to G , then $v \in C(u)$.



Backtracking-based Methods

Matching order generation phase generates the matching order ϕ to guide the enumeration of matched subgraphs.

Definition II.3 (Matching Order). A matching order ϕ is a permutation (i.e., sequence) of query graph's vertex set $V(q)$.



Backtracking-based Methods

The enumeration procedure finds all matches of the query subgraph q in the data graph G with given matching order ϕ .

Definition II.5 (Enumeration Procedure). An enumeration procedure is performed recursively to find subgraph matches f_{iso} with given matching order ϕ and candidate vertex set C .



Subgraph Matching

Subgraph matching has wide applications such as query in graph database and biological graph analytics.

However, it has been proven that the subgraph matching is *NP-complete*. We cannot optimize the worst-case time cost.

In this work, we aim to reduce the enumeration time *on the average case* by proposing a novel query vertex ordering method.



- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments



Limitations of Existing Order Generation Methods

The existing subgraph matching methods usually generate the matching order based on the heuristic values, here are some examples:

- Infrequent edge first ordering
- Infrequent label first ordering
- Path-based ordering.



Limitations of Existing Order Generation Methods

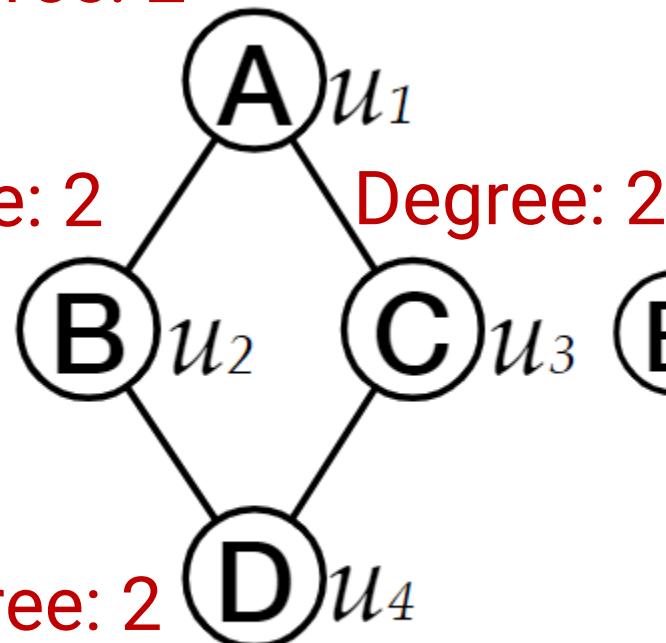
Two major limitations:

- Cannot fully use the graph information.
- Greedy heuristics can lead to local optimum.



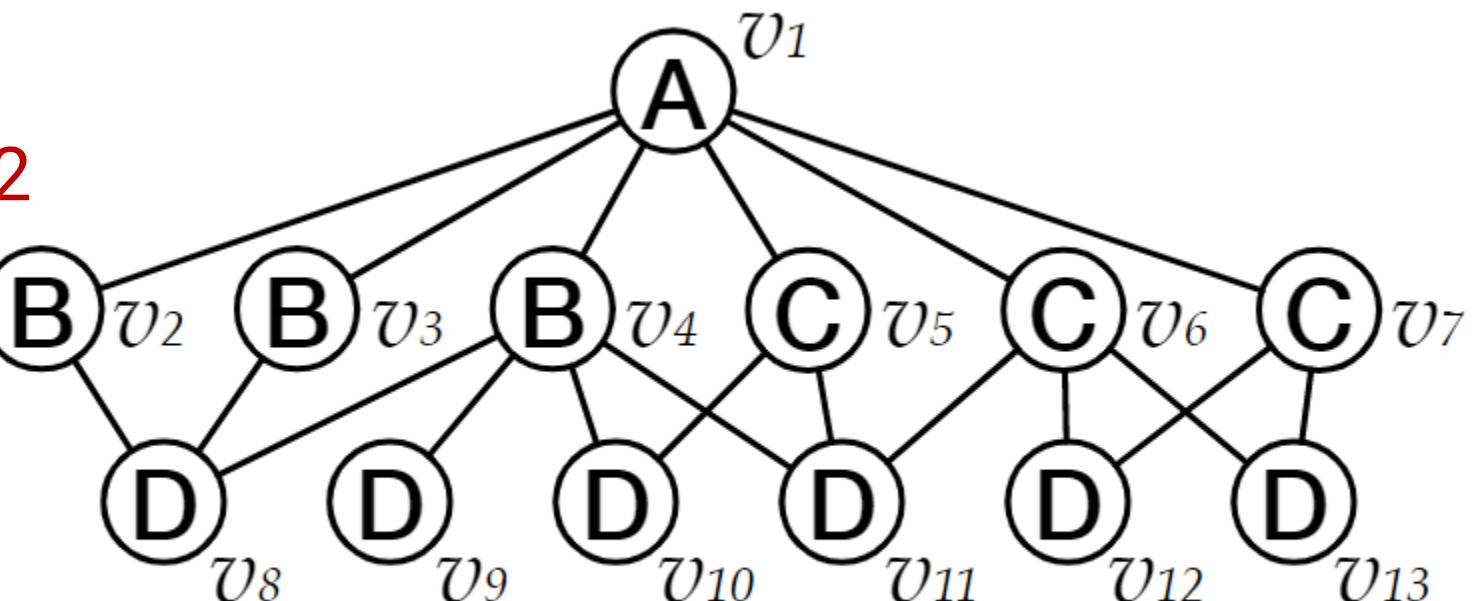
If ordering based on degree (RI)

Degree: 2



(a) Query Graph q

Degree: 2

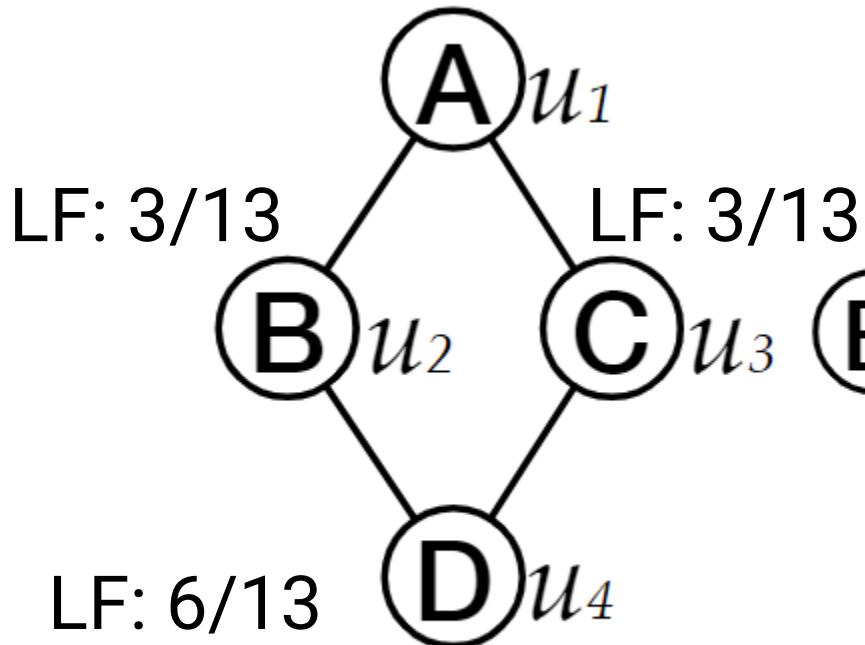


(b) Data Graph G



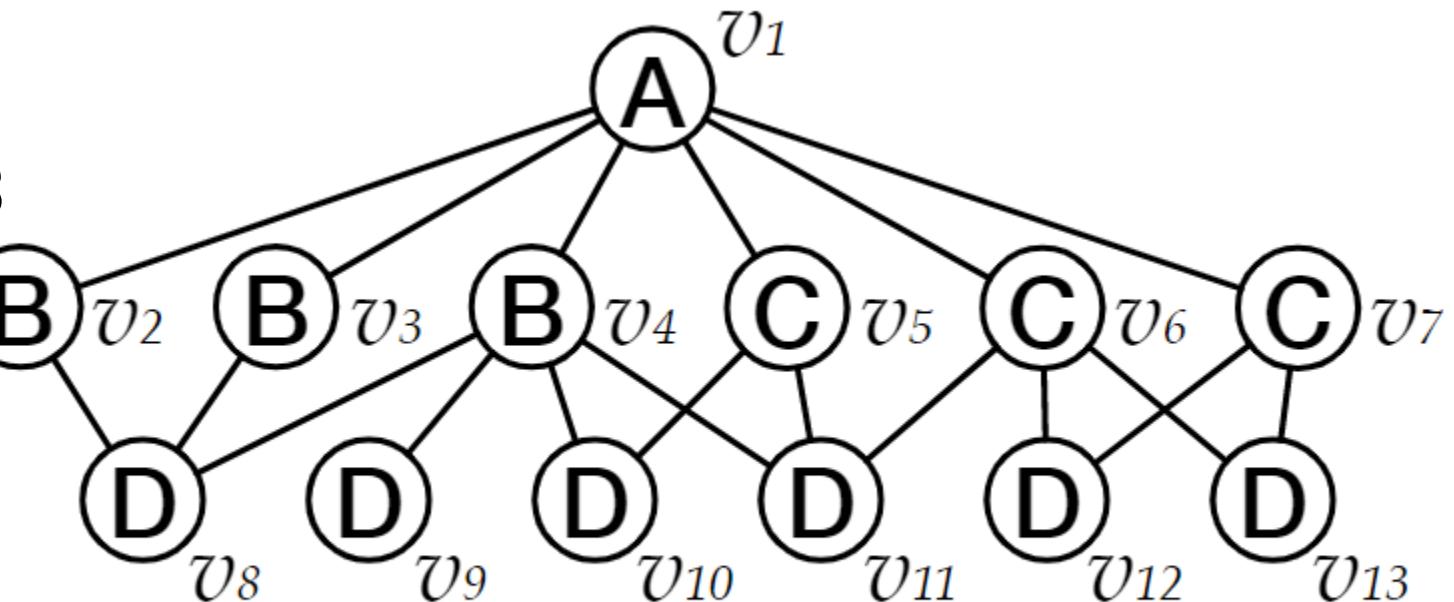
If ordering based on label frequency

LF: 1/13



(a) Query Graph q

LF: Label frequency



(b) Data Graph G



Greedy heuristics can lead to local optimum

The heuristic-based greedy methods can reduce the most redundant intermediate results.

However, these methods cannot consider the long-term query time cost.

The exact optimal order can only be found after all possible order permutations are evaluated.



Our solutions

Motivated by the aforementioned limitations, we proposed the following approaches:

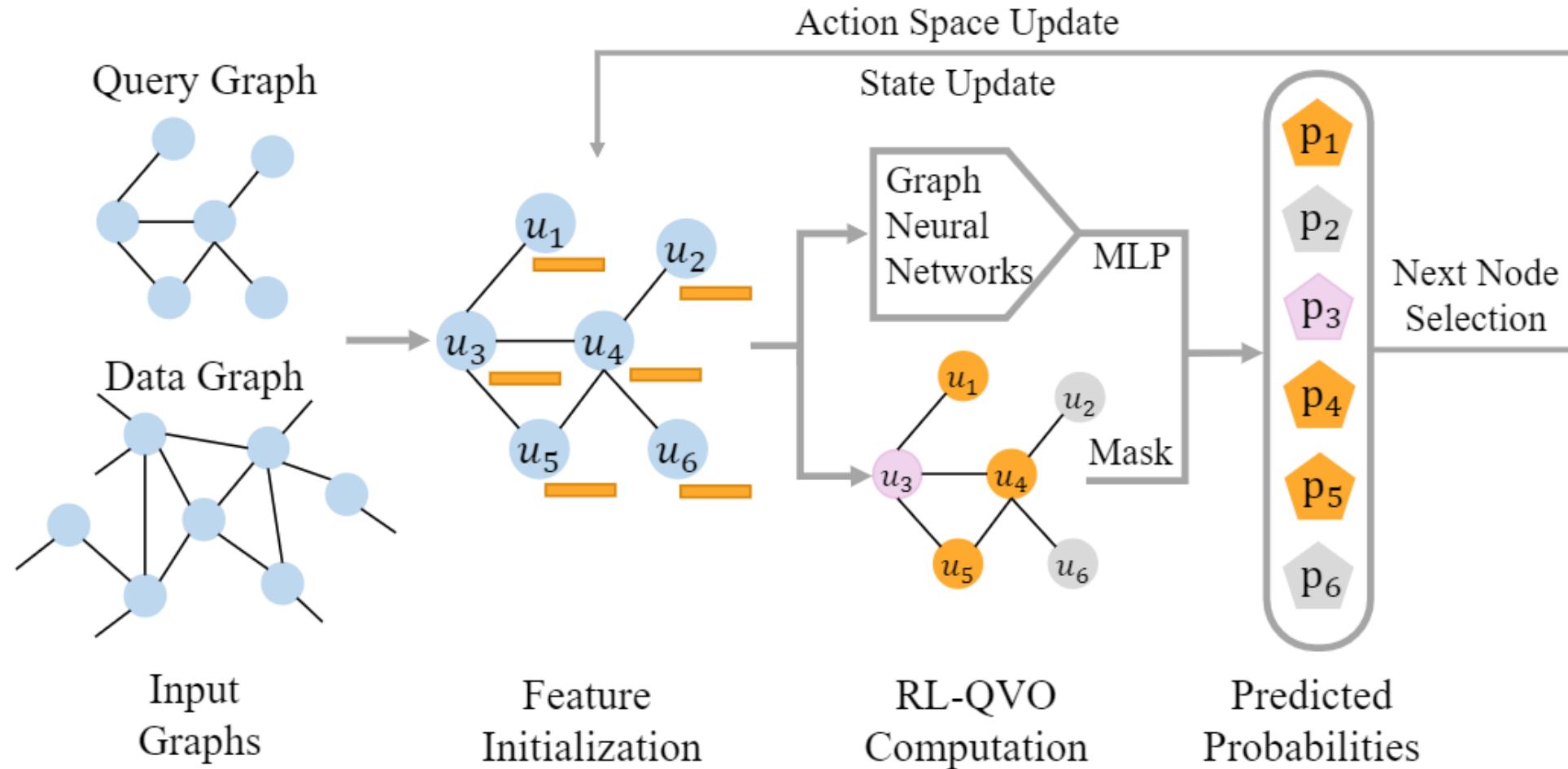
- Capture the graph information with graph neural network.
- Try to approach the global optimal with reinforcement learning.



- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments



Framework



Our recent work

- "Reinforcement learning based query vertex ordering model for subgraph matching" -- ICDE 2022

- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments



Feature Initialization

We use the important statistical heuristics of query vertices to initialize the input query representations.

Based on the input features, our model can fully exploit the information within the features while preserving the relations between the query and data graphs.



Feature Initialization

- Degree of node:

$$\mathbf{h}_u^{(0)}(1) = \text{degree}(u)/\alpha_{\text{degree}},$$

- Label of node:

$$\mathbf{h}_u^{(0)}(2) = \text{label}(u)$$

- Query node ID:

$$\mathbf{h}_u^{(0)}(3) = \text{id}(u)$$



Feature Initialization

- Frequency of data vertices with greater degree than the query node

$$h_u^{(0)}(4) = |\{v \in G | d(u) < d(v)\}| / (|V(G)| \times \alpha_d);$$

- Frequency of data vertex with same label as query vertex:

$$h_u^{(0)}(5) = |\{v \in G | L(u) = L(v)\}| / (|V(G)| \times \alpha_l);$$



Feature Initialization

Lastly, we put two indicator variables in the initial feature:

- Number of unordered vertices:

$$\mathbf{h}_u^t(6) = |V(q)| - t + 1$$

- Trailing indicator that shows whether the node has been ordered:

$$\mathbf{h}_u^t(7) = \mathbb{1}(u \in \phi_{t-1})$$



- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments



Query Vertex Ordering as Markov Decision Process

To exploit the reinforcement learning, we need to model our query vertex ordering problem as a Markov decision process (MDP).

Specifically, we need to define:

- State space
- Action space
- Action probability
- Reward



State Space

In this work, the state space is defined as the all (partial) vertex order sequence.

Specifically, we use ϕ_t (the order at time step t) to denote the state at time step t.



Action Space

With the order ϕ_t (state of the MDP), we define the action space as $N(\phi_t) = \{N(u) | \forall u \in \phi_t, N(u) \notin \phi_t\}$ which is the neighbor vertices set of the ordered vertices in ϕ_t .

Consequently, the action at each time step t is to select a node from the action space (neighbor set $N(\phi_t)$).

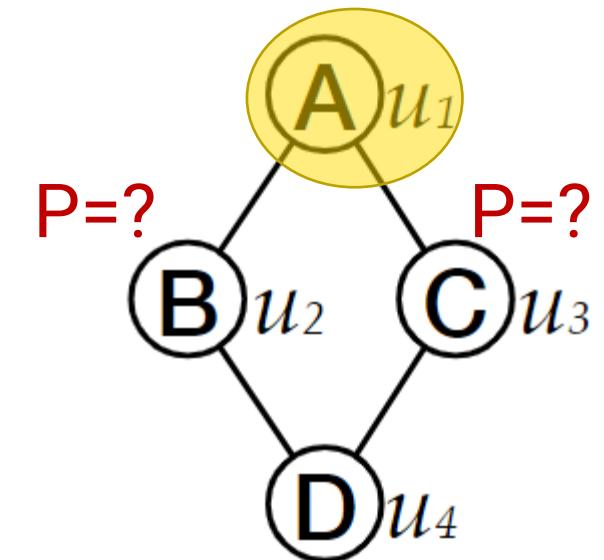


Action Probability

With the state and corresponding action space.

The key problem is to compute the probabilities for each action in the action space.

To this end, we design a Graph Neural Network (GNN)-based policy network.



Policy Network

Graph convolutional network:

$$\mathbf{H}^{(l+1)} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)})$$

Action probability with the state at time step t:

$$\mathbb{P}_{u'}^{(t)} = \pi(\cdot | S^{(t)}) = \text{Softmax}(\text{mask}_{u' \in AS(t)}(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \mathbf{h}_{u'}^{(t)})))$$



Reward Design

Three main rewards:

- Enumeration reward
- Step-wise validate reward
- Entropy reward



Enumeration Reward

Definition II.6 (Enumeration Number). An enumeration number $\#\text{enum}$ is the number of *recursive* calls of the enumeration procedure to find all matches with given q , G , ϕ and C .

The reduced enumeration number:

$$\Delta\#\text{enum} = \#\text{enum}(\phi) - \#\text{enum}(\phi_{\text{base}})$$

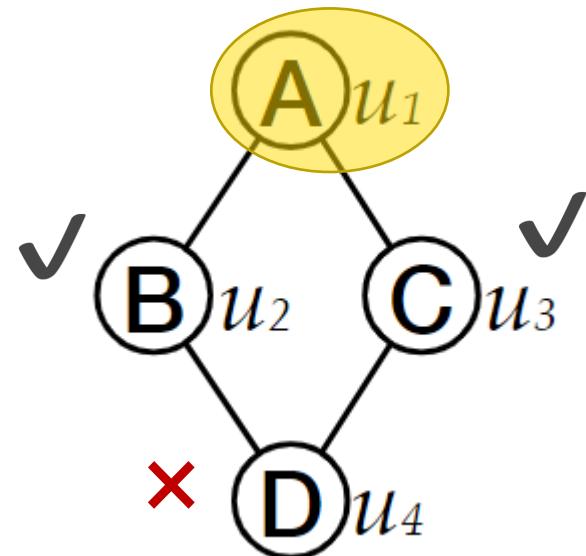
Enumeration reward:

$$r_{\text{enum}} = f_{\text{enum}}(\Delta\#\text{enum}).$$



Other rewards

Step-wise validation: $r_{val,t}$



Entropy reward:

$$r_{h,t} = H(\hat{P}_{\pi_\theta}(\phi_t, N(\phi_t)))$$

Ensure the model can perform more actions.

Reward Design

With the aforementioned rewards, our overall rewards at time step t is as follows:

$$R_t = r_{enum} + \beta_{val} \cdot r_{val,t} + \beta_h \cdot r_{h,t}$$

The overall reward is as follows:

$$R_{q,\theta} = \sum_{t=1}^{|V(q)|} \gamma^t R_t,$$



- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments

Policy Training

In this work, we use the proximal policy optimization (PPO) to train our policy network with the following loss function:

$$J_r^{(t)}(\theta) = \sum_{(a_t, s_t)} \min\left(\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta'}(a_t | s_t)} r_t(\theta), \text{clip}\left(\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta'}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon\right) r_t(\theta)\right)$$

$$J(\theta) = \sum_{t=1}^{|V(q)|} J_r^{(t)}(\theta)$$



- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments

Experiment Setup

Compared methods:
QuickSI
RI
VF2++
VEQ
Hybrid
RL-QVO

Dataset Statistics

Dataset	V	E	L	d
Citeseer	3,327	4,732	6	1.4
Yeast	3,112	12,519	71	8.0
DBLP	317,080	1,049,866	15	6.6
Youtube	1,134,890	2,987,624	25	5.3
Wordnet	76,853	120,399	5	3.1
EU2005	862,664	16,138,468	40	37.4

Evaluation Metrics

- Query processing time
- Enumeration time



Average Query Processing Time

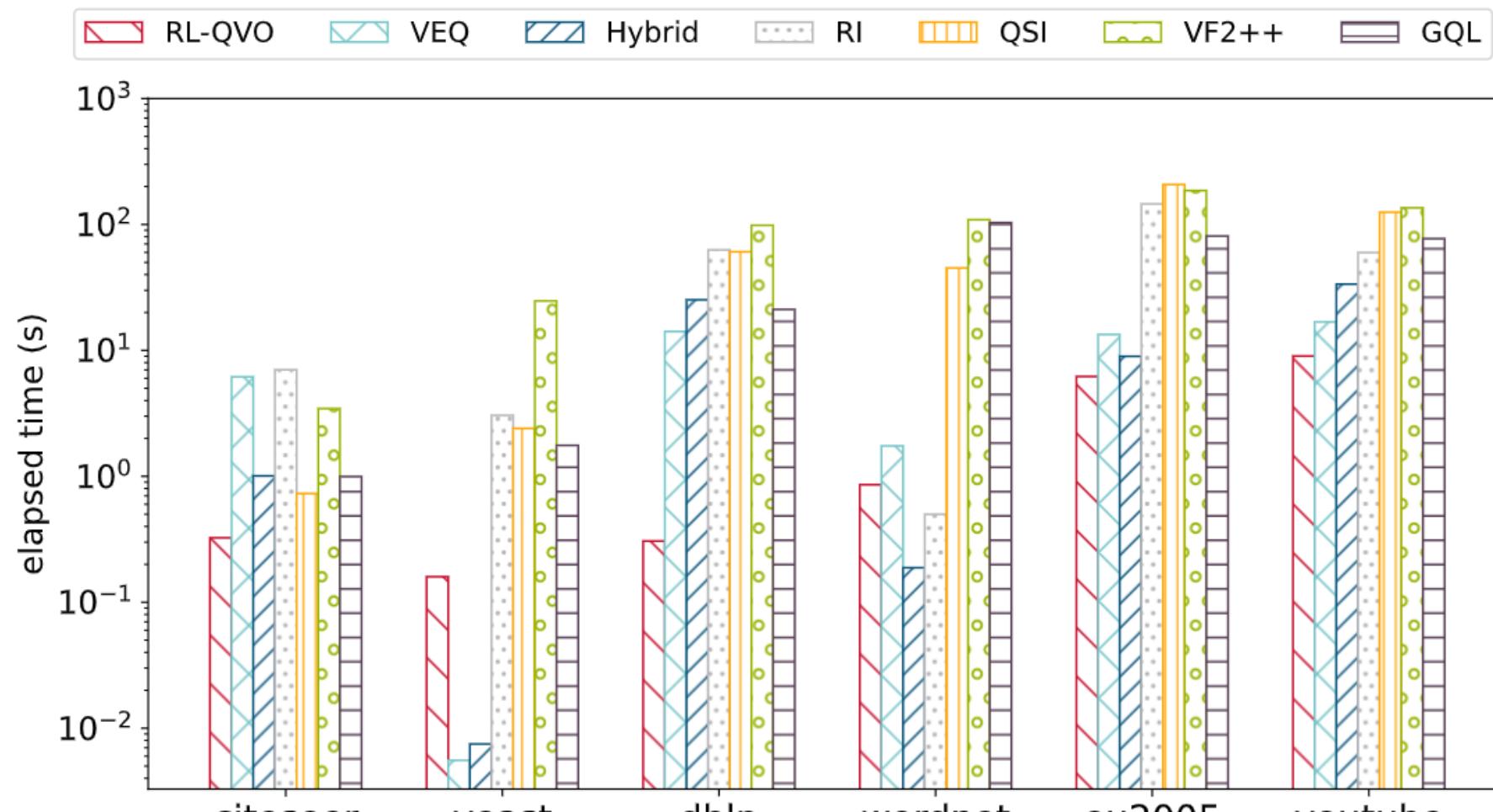
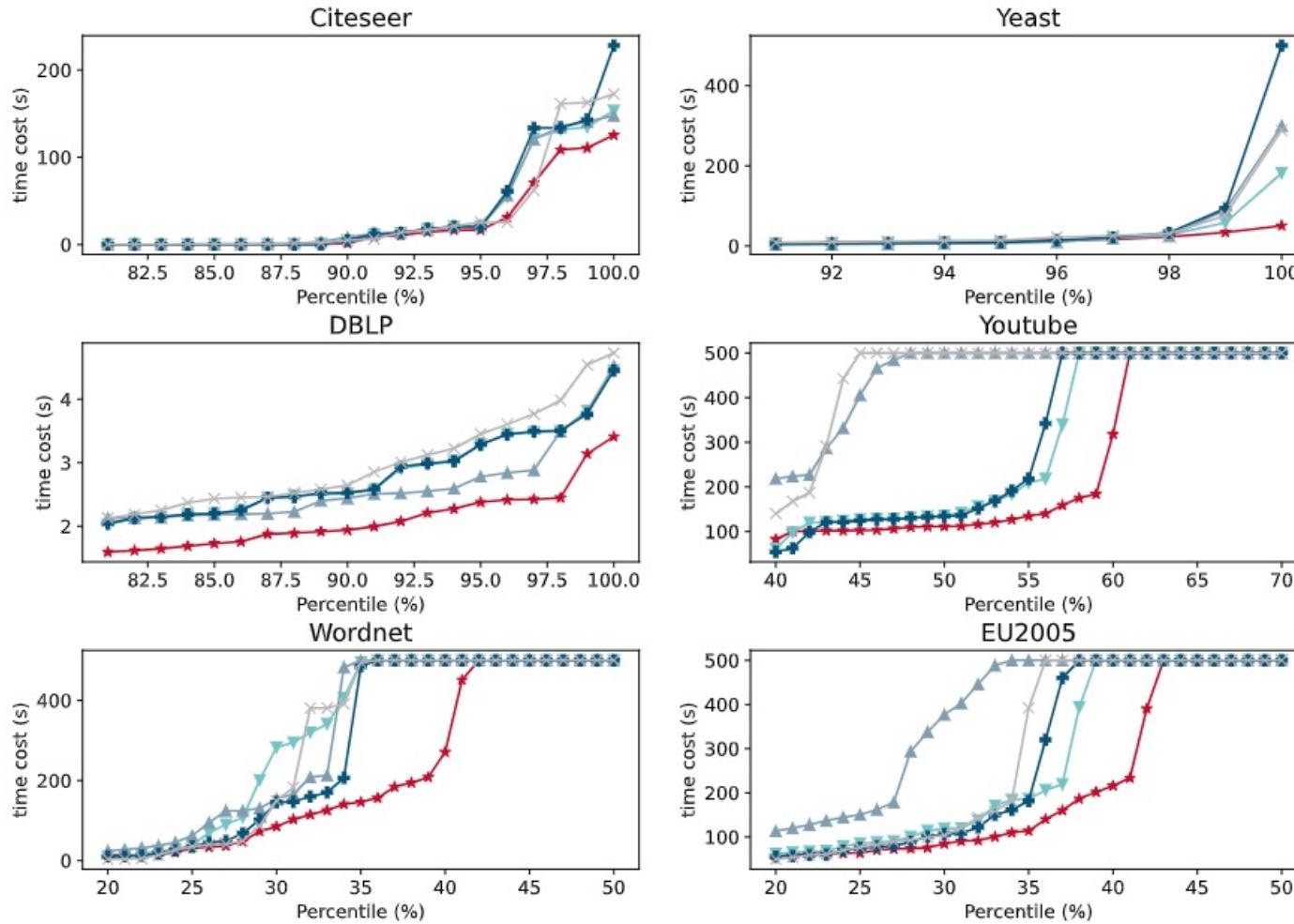
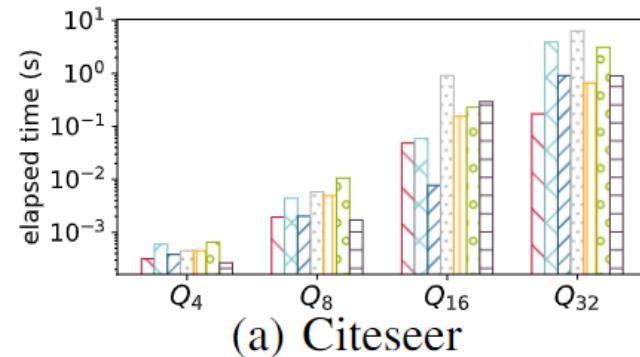


Fig. 3: Average Query Processing Time Comparison

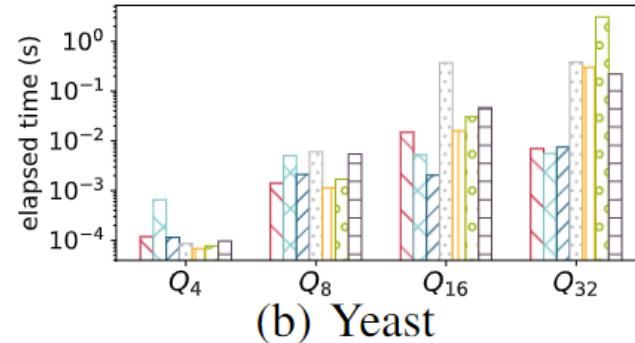
Query Processing Time Percentile Comparison



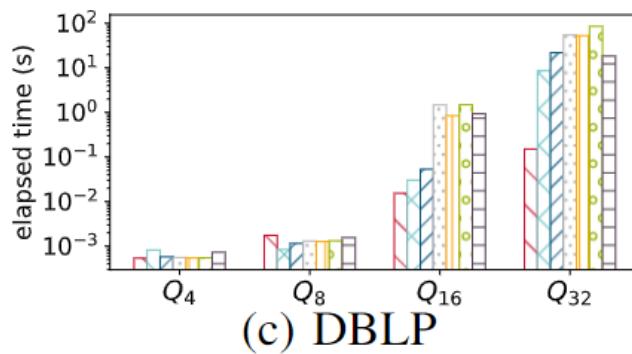
Enumeration Time Comparison



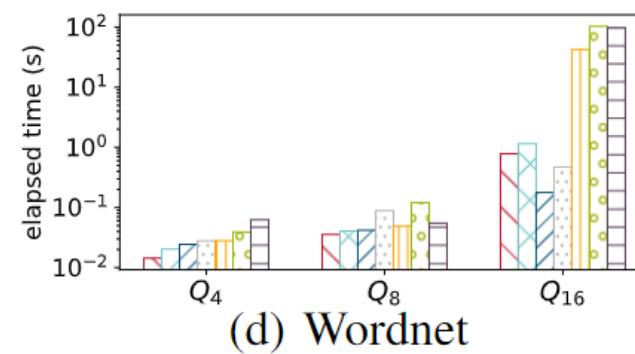
(a) Citeseer



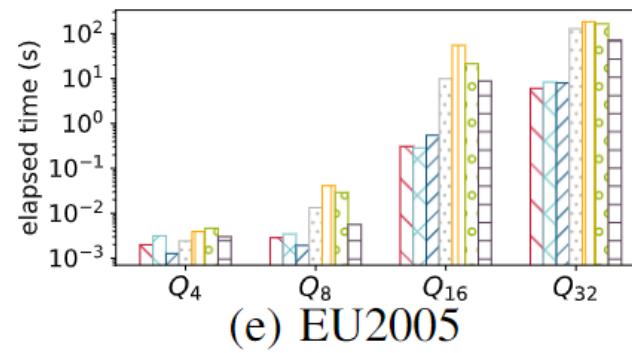
(b) Yeast



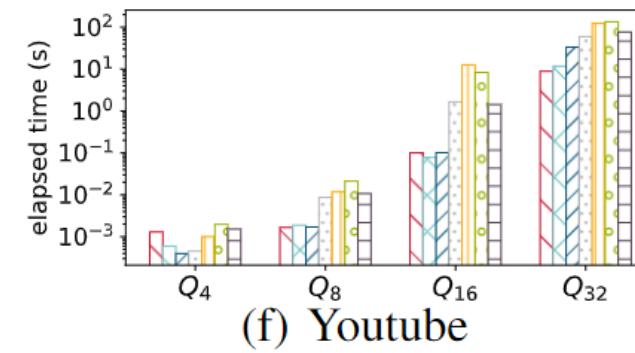
(c) DBLP



(d) Wordnet



(e) EU2005



(f) Youtube

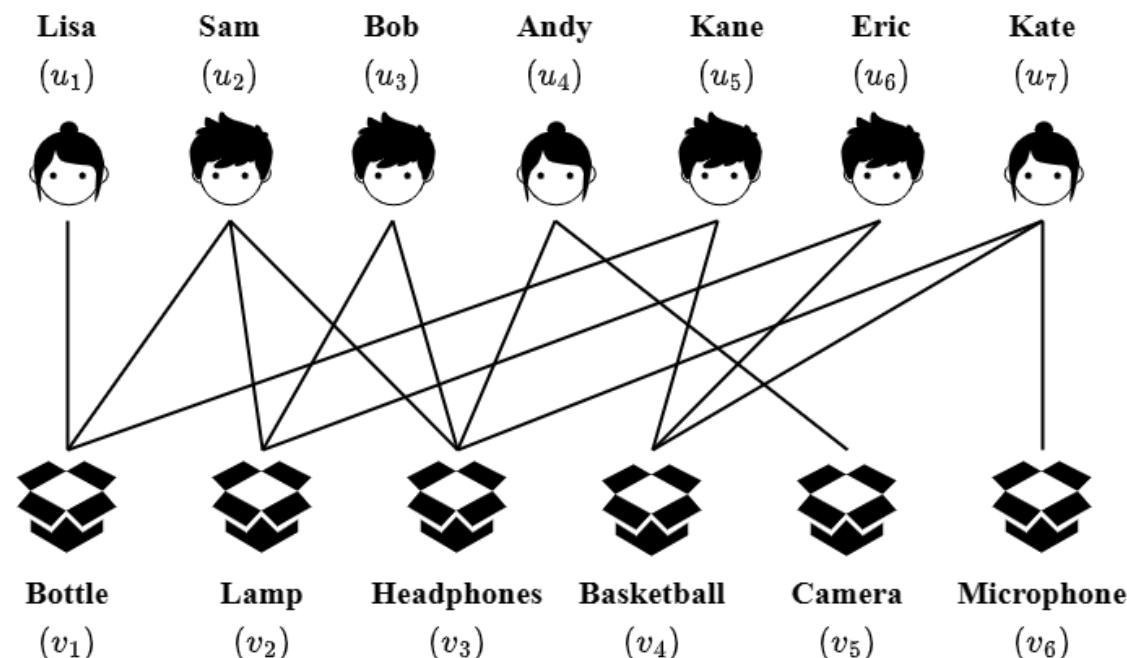
Fraud Detection



Background

Attributed Bipartite Graph

An attributed bipartite graph is a type of graph which consists of two sets of vertices that are linked by edges. The vertices have additional attributes, making this graph particularly useful for **representing information in the field of e-commerce**.



Background

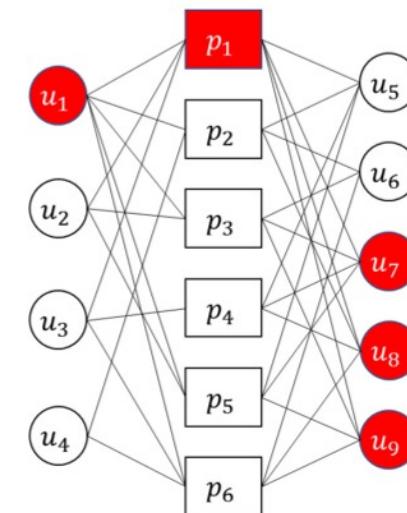
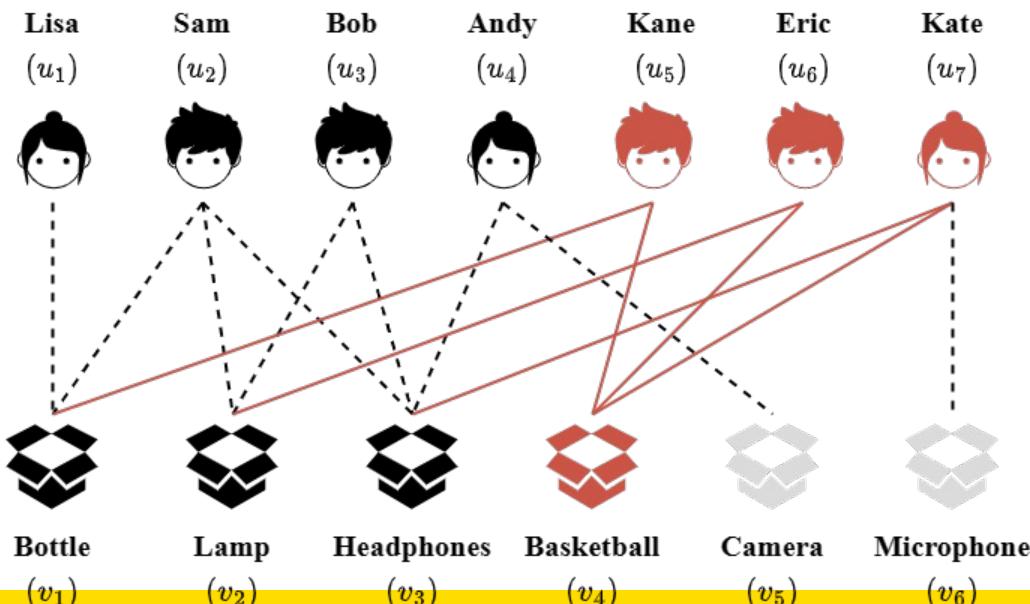
Group-based Frauds on Attributed Bipartite Graphs

Group-based fraud is becoming increasingly common:

“Ride Item’s Coattails” attack (edge classification)

Sockpuppet-based Targeted Attack on Reviewing Systems

(STARS attack) (vertex classification)



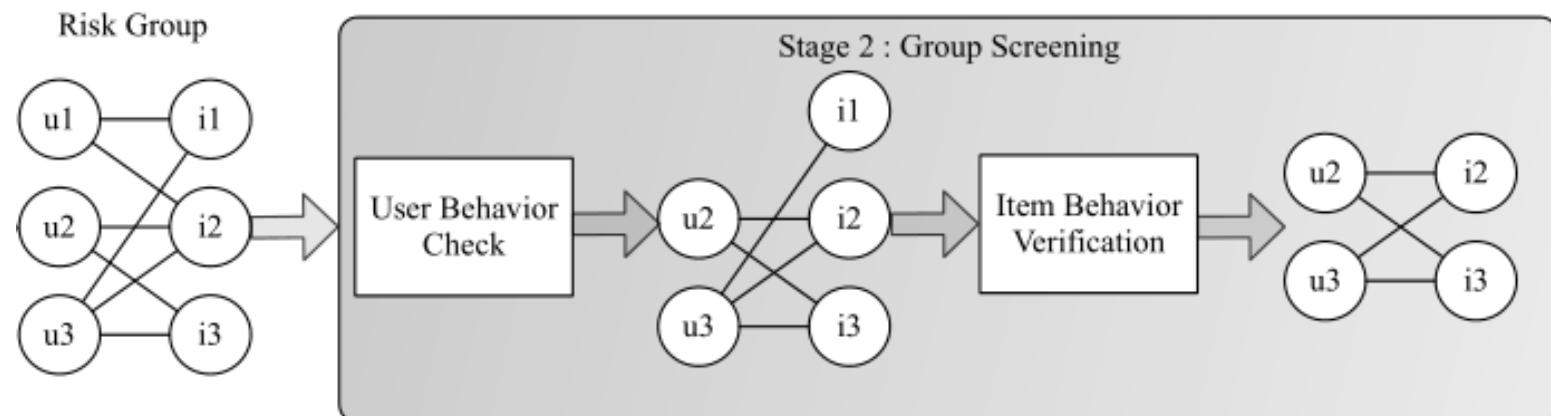
Rating	Score	Rating	Score
(u_1, p_1)	1	(u_5, p_5)	0.5
(u_1, p_2)	-1	(u_6, p_2)	0
(u_1, p_3)	0.5	(u_6, p_3)	1
(u_1, p_5)	0.5	(u_6, p_6)	1
(u_1, p_6)	0.5	(u_7, p_1)	1
(u_2, p_1)	0	(u_7, p_3)	1
(u_2, p_3)	1	(u_7, p_4)	1
(u_2, p_5)	1	(u_7, p_6)	0.5
(u_3, p_1)	0	(u_8, p_1)	1
(u_3, p_4)	-0.5	(u_8, p_2)	0
(u_3, p_6)	0.5	(u_8, p_4)	0.5
(u_4, p_2)	-1	(u_8, p_6)	1
(u_4, p_6)	1	(u_9, p_1)	1
(u_5, p_1)	-1	(u_9, p_3)	0.5
(u_5, p_2)	-1	(u_9, p_5)	0.5
(u_5, p_4)	0	(u_9, p_6)	0.5

Background

SOTA method for “Ride Item’s Coattails” attack

RICD ((α, k_1, k_2) -biclique): **fraud detection method** for “Ride Item’s Coattails” attack. Can only utilize structural information.

Tianchi competition winner’s algorithm: **classification method**. Can only use attribute information.



Background

Existing methods

Classification Methods:

- Imbalanced labeled vertices, community information.

Cohesive Subgraph Mining Methods:

- Attribute and label information, suffer from NP-completeness.

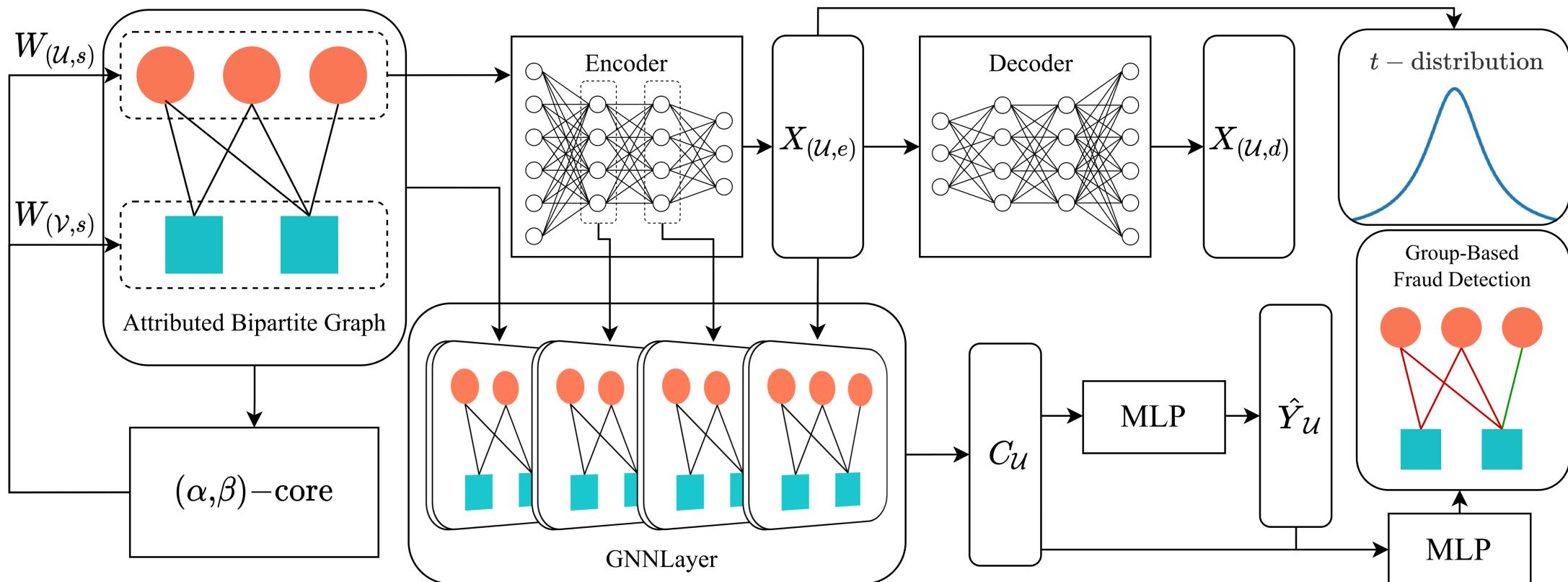
Fraud Detection Methods:

- Global topological and attribute information, label information, manual parameter setting.



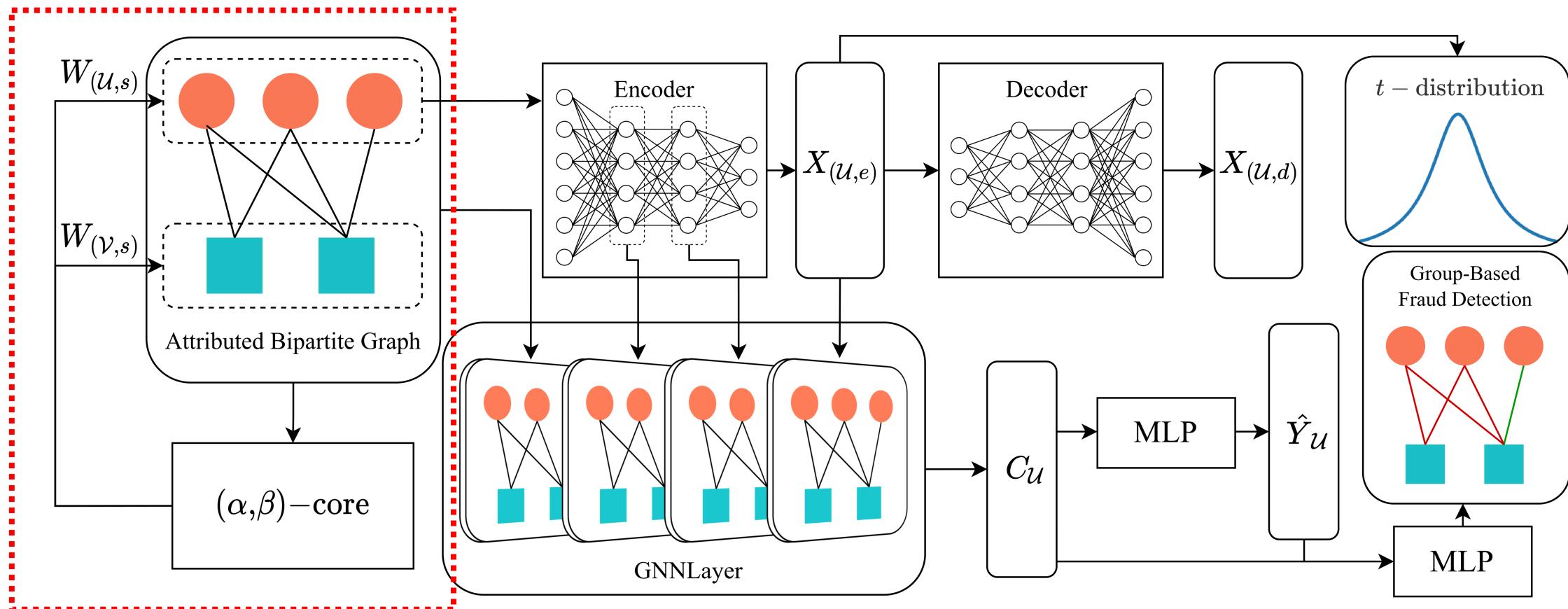
Overview

Group-based Fraud Detection method: GFDN



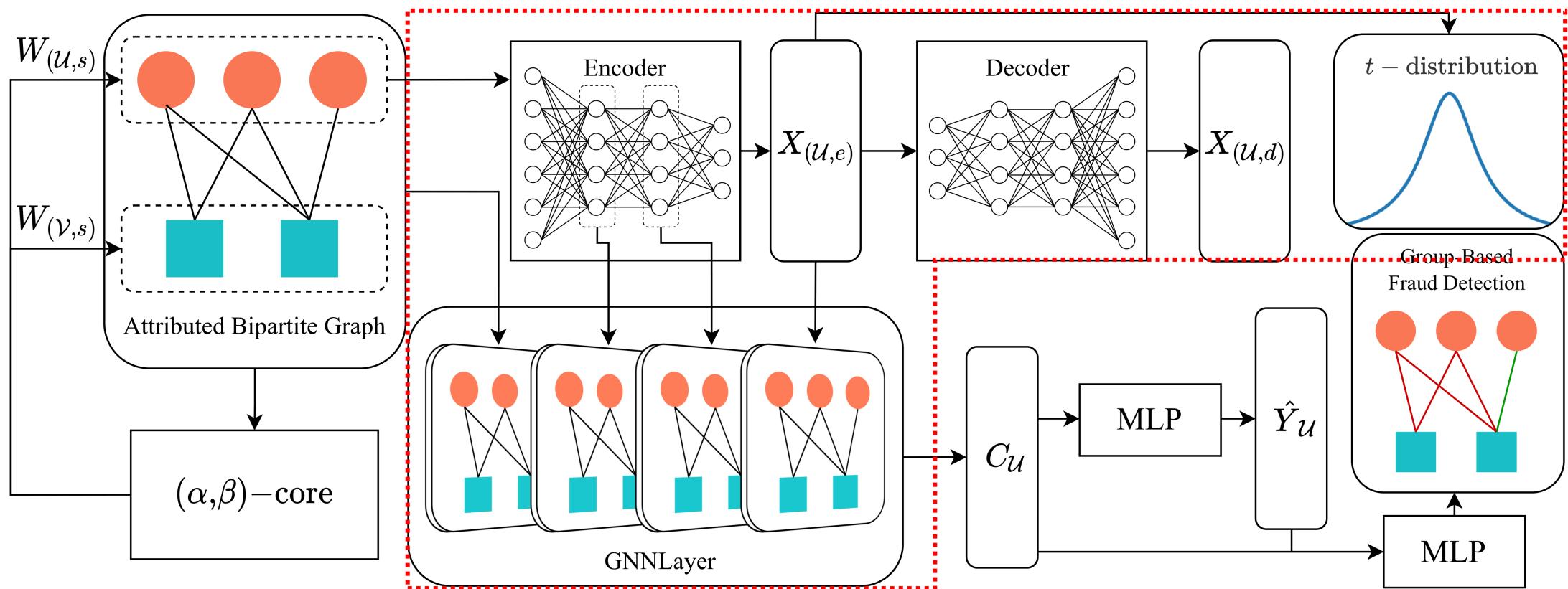
Overview

Group-based Fraud Detection method: GFDN



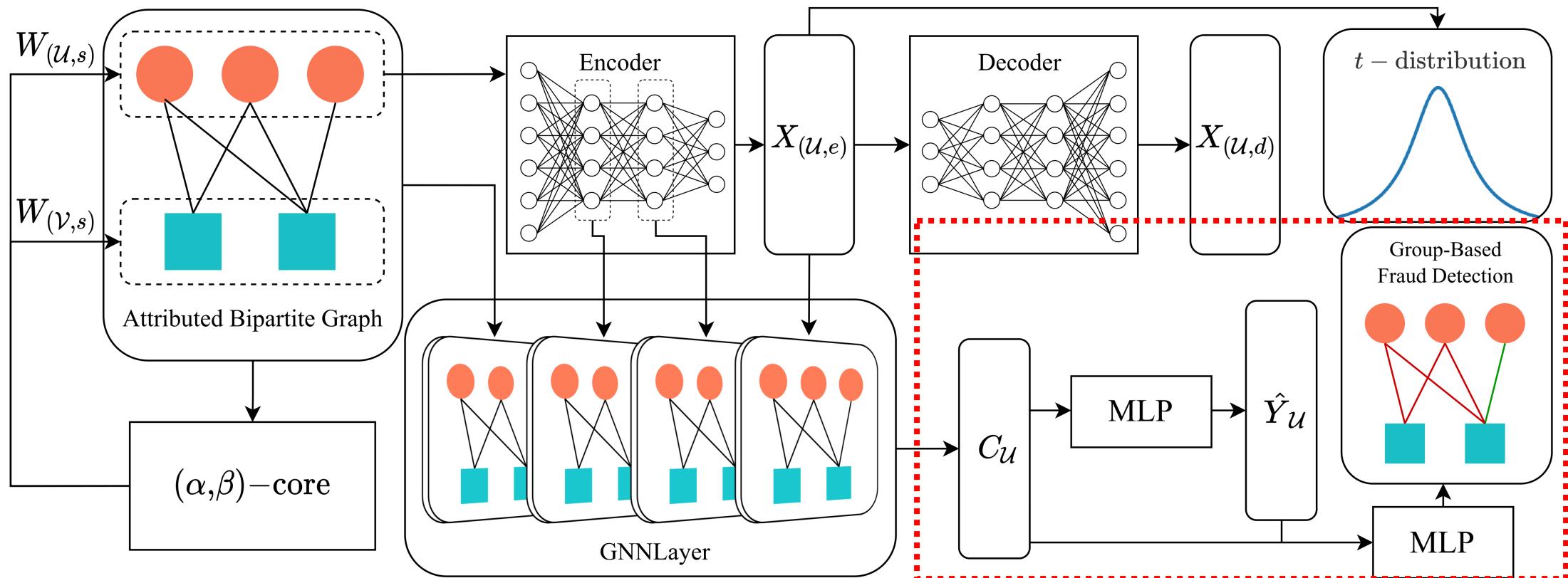
Overview

Group-based Fraud Detection method: GFDN



Overview

Group-based Fraud Detection method: GFDN

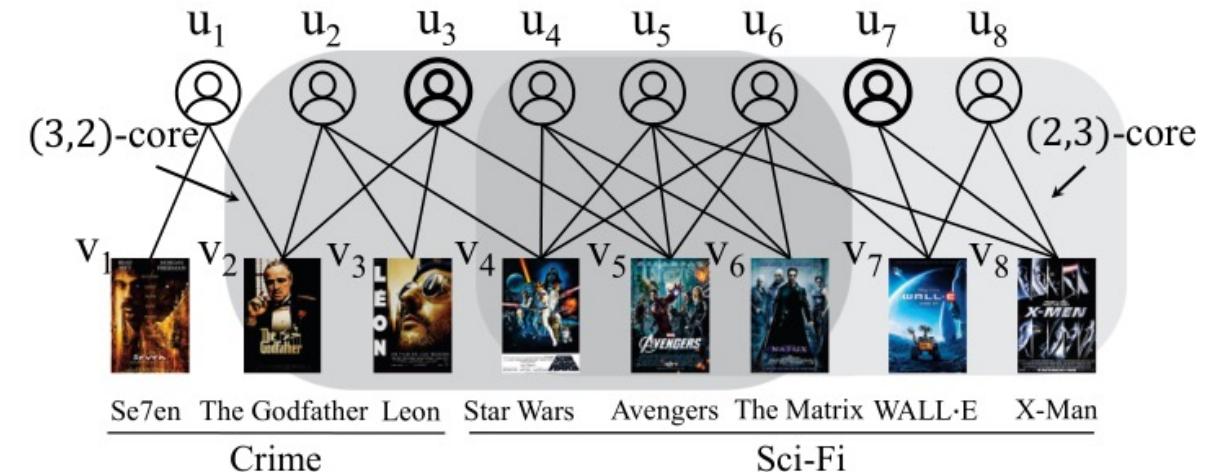


Structural Feature Initialization

(α, β) -core:

Given a bipartite graph G and integers $\alpha, \beta \in \mathbb{Z}^+$, (α, β) -core of G is denoted as G' which consists of two vertex sets $U' \subseteq U$ and $V' \subseteq V$.

The (α, β) -core G' is a maximal bipartite subgraph induced by $U' \cup V'$ from G in which all the vertices in U' have degrees at least α and all the vertices in V' have degrees at least β .



GFDN

Structural Feature Initialization

GFDN will generate structural features for vertices based on their existence in different (α, β) -core.

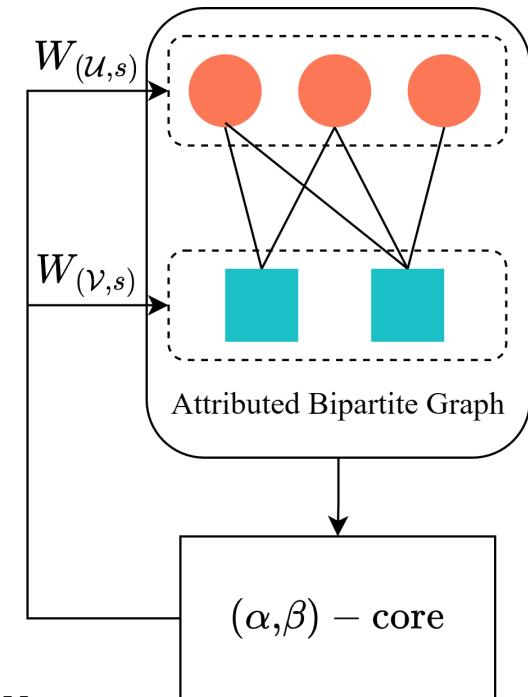
$$\hat{X}_{(\mathcal{U},s)} = X_{(\mathcal{U},s)} \odot (I_{\mathcal{U}} W_{(\mathcal{U},s)}), \quad \hat{X}_{(\mathcal{V},s)} = X_{(\mathcal{V},s)} \odot (I_{\mathcal{V}} W_{(\mathcal{V},s)})$$

Structural Features

Element-wise Product

All-ones Vector

Weight Matrix



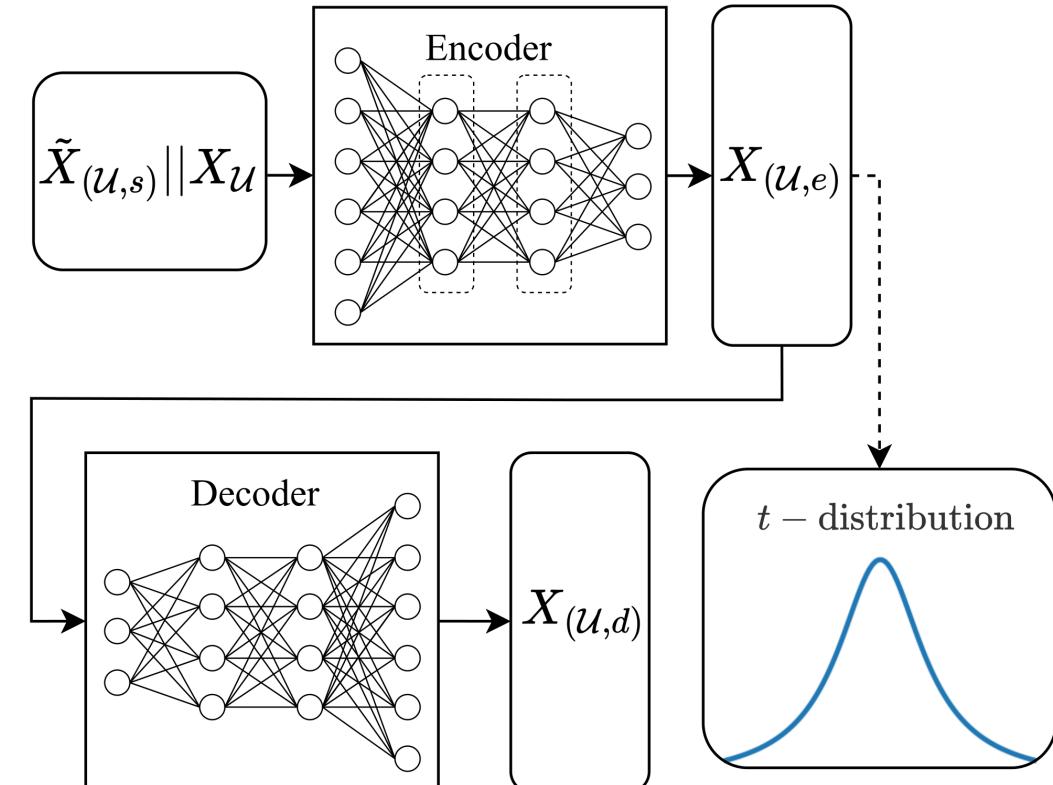
Fraudster Community Detection

BDCN - Autoencoder:

Autoencoder in Bipartite Deep Clustering Network (BDCN) can:

1. preserving both structural and attribute information from the input features.
2. Generate high-quality community representation for customer vertices.

It can achieve self-supervised fraud **community detection** using a loss function measures with Student's t-distribution kernel.

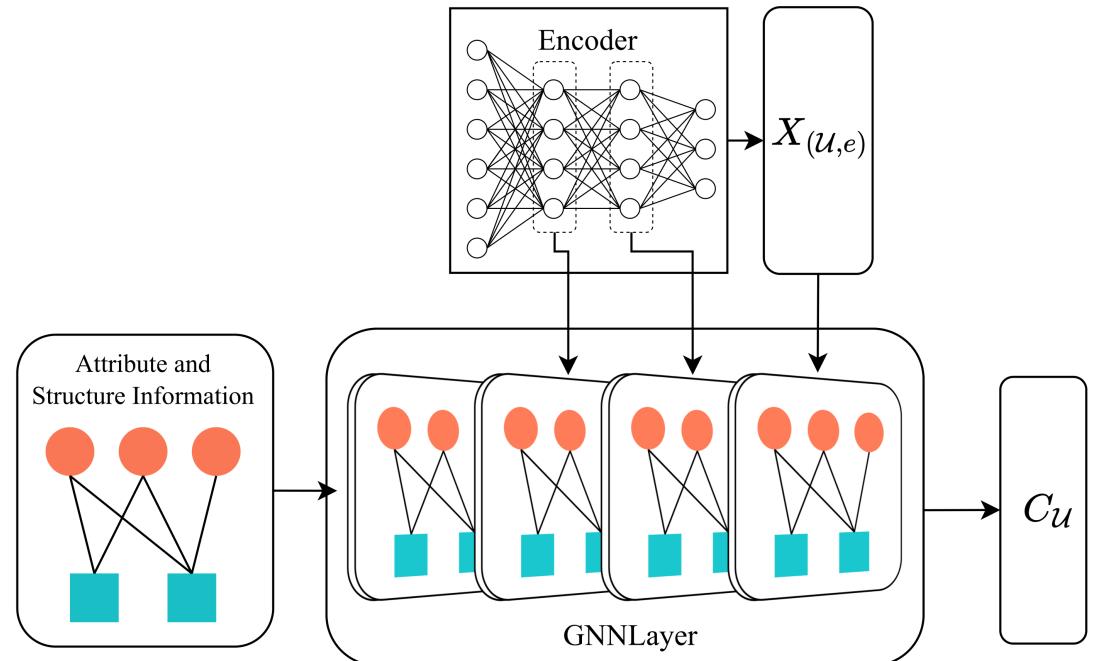


GFDN

Fraudster Community Detection

BDCN - GNN:

GNN in BDCN can aggregate on attribute bipartite graph and preserve the attribute information and structural information of the graph. The output of encoding layer will be used.



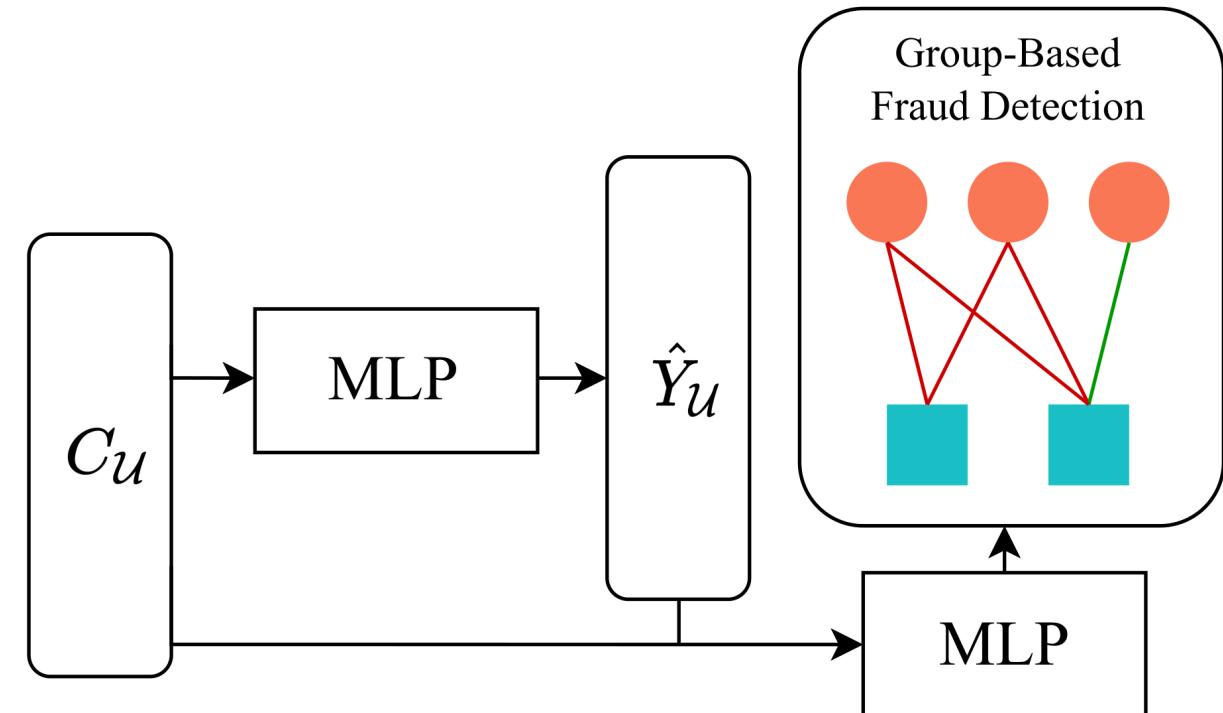
Training Objective

"Ride Item's Coattails" Attack:

In "Ride Item's Coattails" attack, not all edges related to fraudsters necessarily have attack implications. GFDN will perform **multi-task training** on this issue, predicting both **fraudsters** and **fraudulent attack**.

STARS Attack:

STARS attack detection aims to **detect fraudsters**, in which case GFDN only needs to perform the vertex classification task.



Training Objective

The final loss function will be composed of the loss functions of the aforementioned modules, including **autoencoder**, **community prediction**, **fraudster prediction**, and **fraudulent prediction**. The sum of the weights of all parts of them is 1.

$$\mathcal{L} = \omega_{ae} \mathcal{L}_{ae} + \omega_c \mathcal{L}_c + \omega_l \mathcal{L}_l + \omega_e \mathcal{L}_e$$

Autoencoder Community Fraudster Fraudulent

Experiments

Experimental Setup

- **Dataset**
 - 4 real-life datasets.
- **Compared methods**
 - 5 learning-based methods.
 - 2 pattern-based methods.
 - 4 fraud detection methods.
 - A naïve model and four ablated GFDNs
- **Parameter settings**
 - The number of GNN layer: 4.
 - The number of community: 32.
 - Hidden dimension: 128.
 - The selected GNN is GraphSAGE.
- **Implementation**
 - Structure information extraction: C++
 - Other Parts of the Model :Python + Pytorch Geometric.

Table 1: Datasets for “Ride Item’s Coattails” Attack Detection

Dataset	$ \mathcal{E} $	$ \mathcal{U} $	$ \mathcal{V} $	% Fraudulent	% Legitimate
TB	3,085,653	996,090	381,611	0.62%	3.53%
TC	1,050,000	532,345	239,840	2.86%	11.43%

Table 2: Datasets for STARS Attack Detection

Dataset	$ \mathcal{E} $	$ \mathcal{U} $	$ \mathcal{V} $	% Fraudulent	% Legitimate
Alpha	24,186	3,286	3,754	3.10%	4.20%
OTC	35,592	4,814	5,858	3.70%	2.80%

Experiments

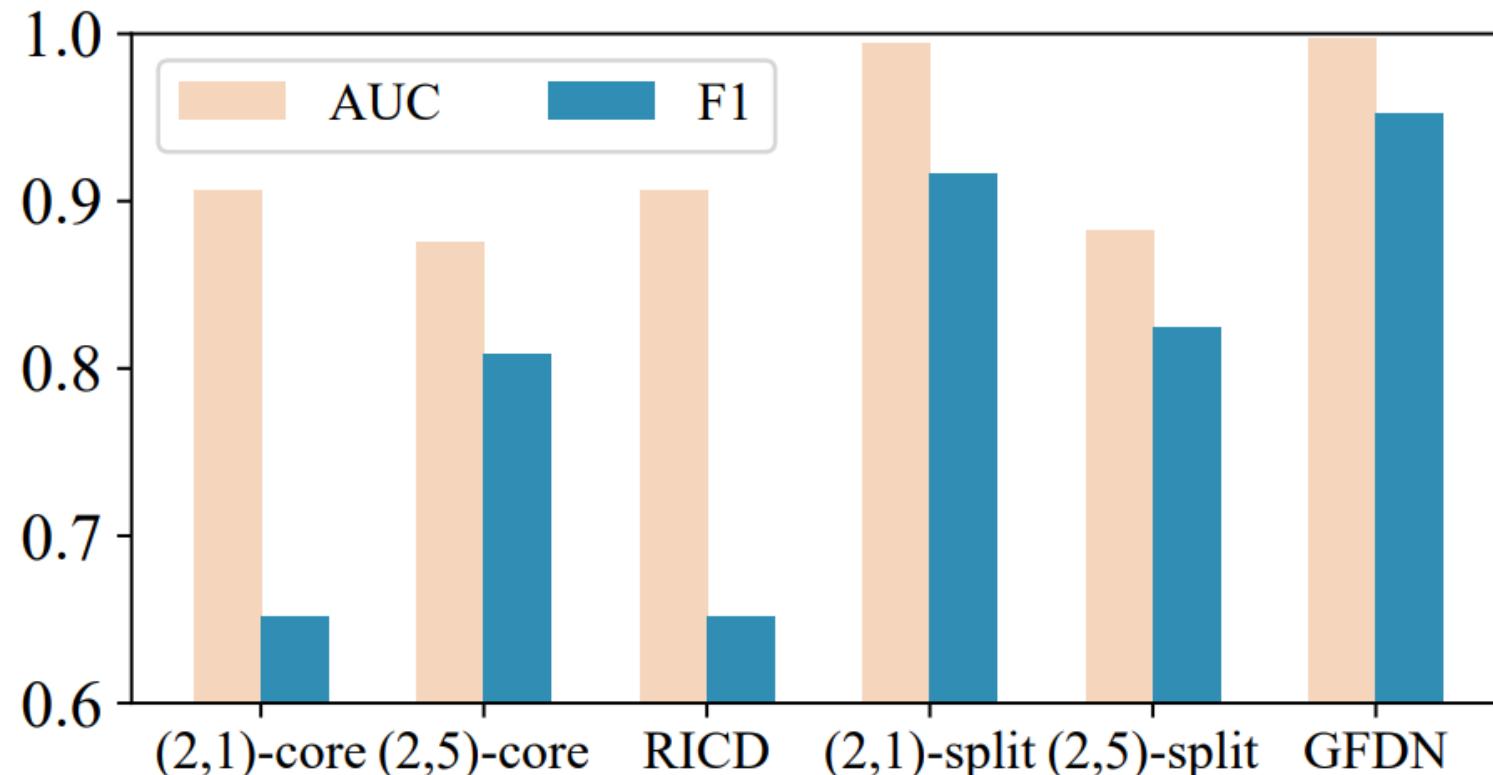
Effectiveness Evaluation Results for “Ride Item’s Coattails” Detection

	TB Data					TC Data				
	F1	Acc	AUC	Pre	Recall	F1	Acc	AUC	Pre	Recall
LPA	0.2737	0.4627	0.5517	0.1715	0.6785	0.2056	0.4284	0.5276	0.1219	0.6557
SBGNN	0.4789	0.8228	0.7947	0.4279	0.5438	0.3676	0.8074	0.7666	0.2900	0.5018
BiGI	0.5359	0.8540	0.8491	0.5097	0.5649	0.4039	0.8292	0.8044	0.3331	0.5129
SIHG	0.6449	0.8709	0.8692	0.5470	0.7853	0.5947	0.8771	0.8985	0.4735	0.7992
Tianchi	0.6446	0.8752	0.9342	0.5606	0.7581	0.5364	0.8717	0.9107	0.4527	0.6583
RICD	0.6518	0.8405	0.9063	0.4834	1.0000	0.4784	0.8482	0.7474	0.3906	0.6171
(α, β) -core	0.8081	0.9449	0.8757	0.8417	0.7770	0.6348	0.8907	0.8696	0.5093	0.8423
FRAUDAR	0.2580	0.1481	0.4963	0.1483	0.9927	0.2020	0.1124	0.4981	0.1124	0.9961
CF1	0.2407	0.7698	0.5532	0.2371	0.2445	0.1620	0.7981	0.5253	0.1523	0.1731
CF2	0.4675	0.7603	0.7376	0.3497	0.7052	0.3588	0.6837	0.7277	0.2326	0.7844
Naive	0.8109	0.9473	0.9844	0.8736	0.7565	0.6397	0.9090	0.9516	0.7816	0.5414
GFDN-S	0.6867	0.9202	0.9653	0.8284	0.5864	0.6122	0.8783	0.9342	0.4780	0.8514
GFDN-F	0.9212	0.9754	0.9886	0.8821	0.9639	0.6401	0.8976	0.9287	0.5302	0.8076
GFDN-L	0.9398	0.9813	0.9964	0.9050	0.9775	0.7015	0.9192	0.9654	0.6014	0.8417
GFDN-C	0.9423	0.9821	0.9967	0.9086	0.9785	0.7048	0.9226	0.9646	0.6181	0.8198
GFDN	0.9522	0.9853	0.9974	0.9254	0.9806	0.7226	0.9242	0.9713	0.6154	0.8752



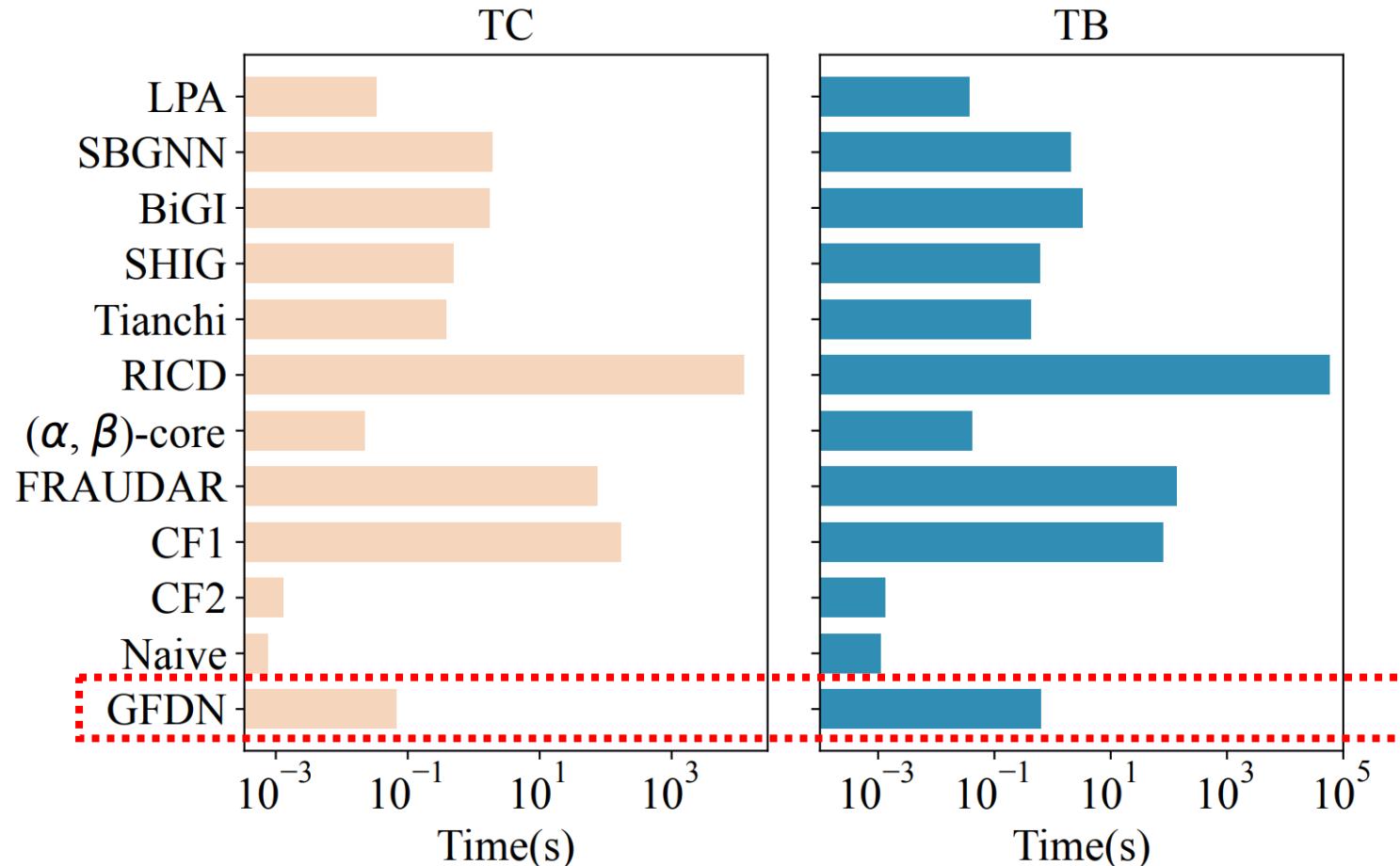
Experiments

Comparison with Pattern-based Algorithms



Experiments

Query Time Evaluation of “Ride Item’s Coattails” Detection



Experiments

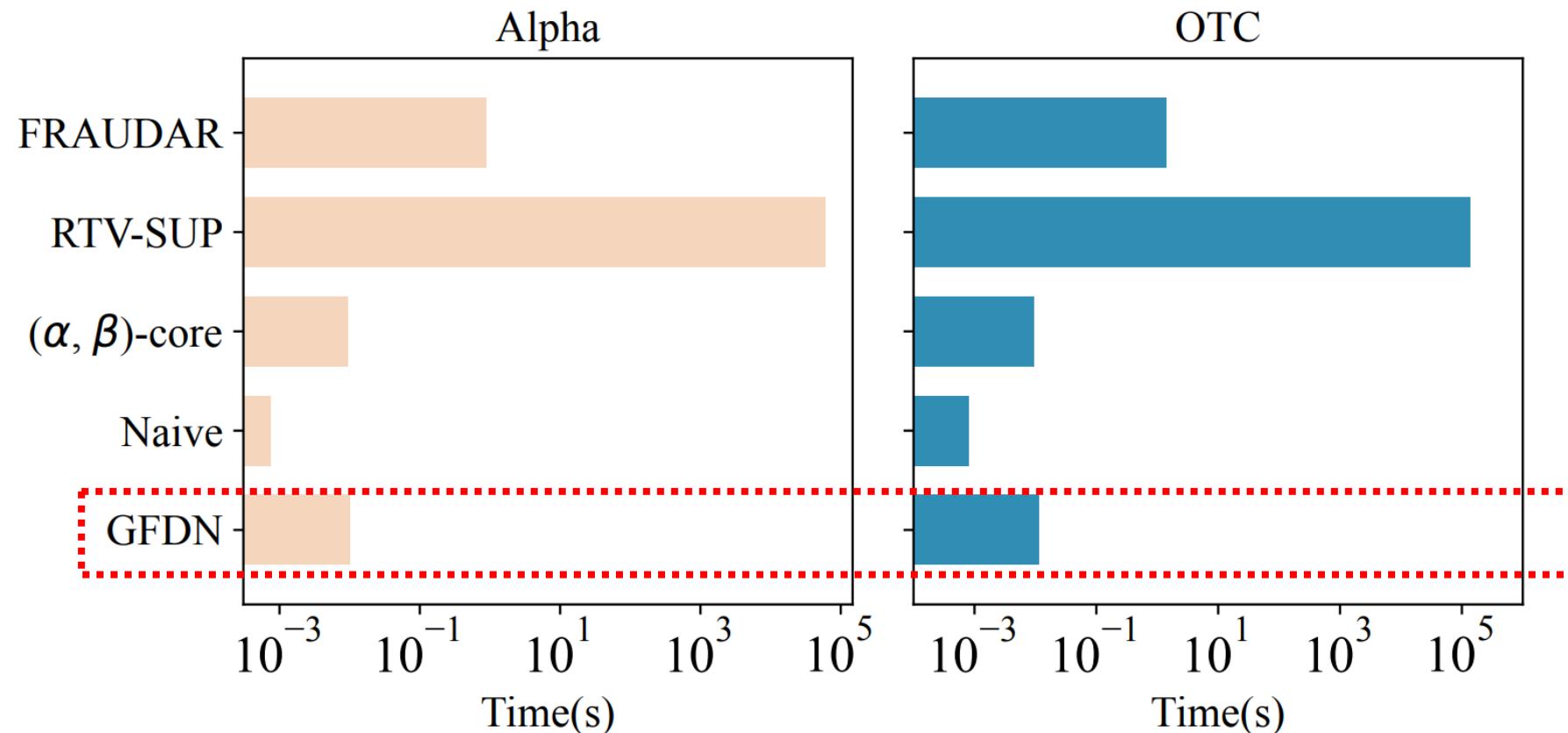
Effectiveness Evaluation Results for STARS Detection

	Alpha					OTC				
	F1	Acc	AUC	Pre	Recall	F1	Acc	AUC	Pre	Recall
FRAUDAR	0.3800	0.2626	0.5236	0.2346	1.0000	0.3780	0.2547	0.5183	0.2330	1.0000
RTV-SUP	0.8652	0.9452	0.8859	0.9747	0.7778	0.7010	0.8082	0.8736	0.5417	0.9931
(α, β) -core	0.7857	0.8767	0.9204	0.6471	1.0000	0.7784	0.8711	0.9167	0.6372	1.0000
Naive	0.8089	0.9018	0.9789	0.7222	0.9192	0.7937	0.8978	0.9508	0.7310	0.8681
GFDN	0.8919	0.9452	0.9913	0.8049	1.0000	0.9231	0.9623	0.9746	0.8571	1.0000



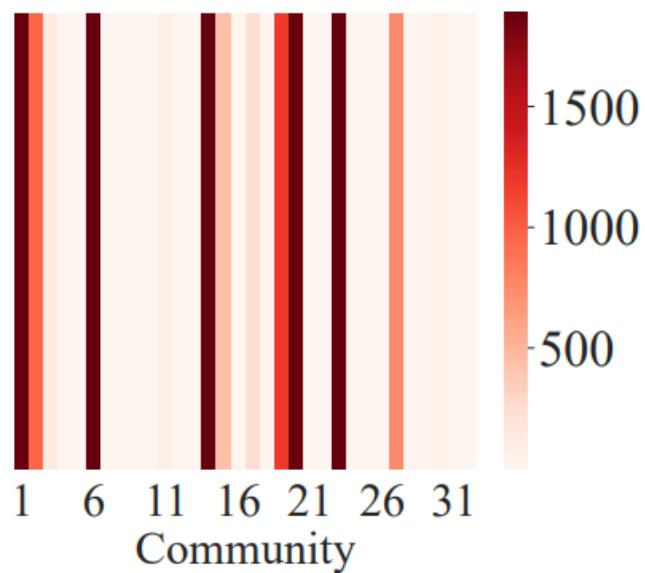
Experiments

Efficiency Evaluation Results for STARS Detection

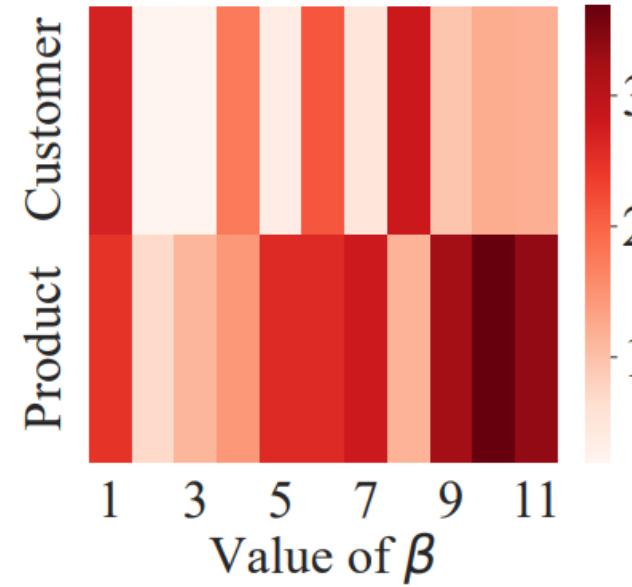


Experiments

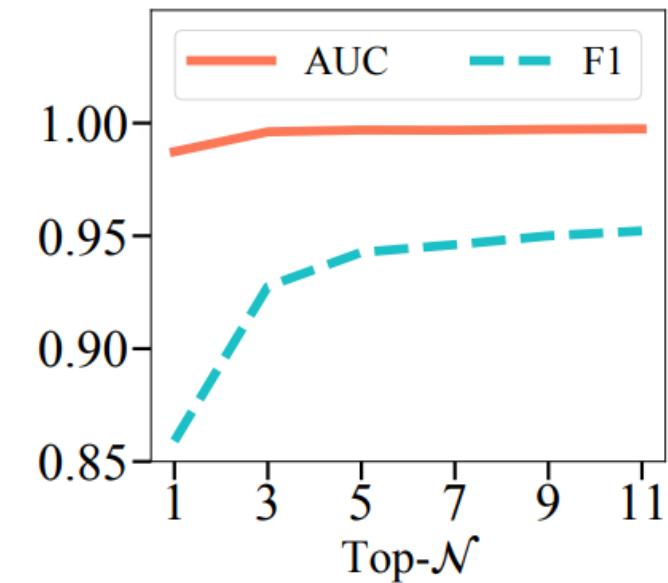
In-Depth Effectiveness Analysis of GFDN



(a) Heatmap of C_U



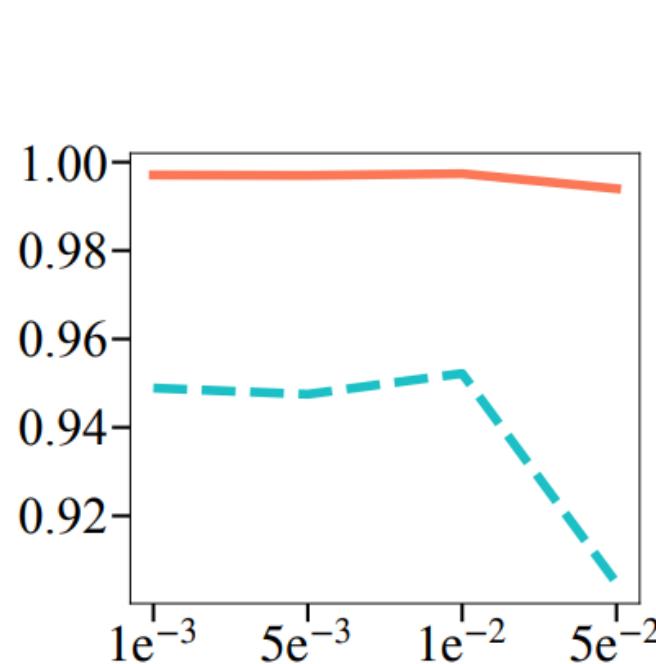
(b) Heatmap of $W_{(U,s)}$, $W_{(V,s)}$



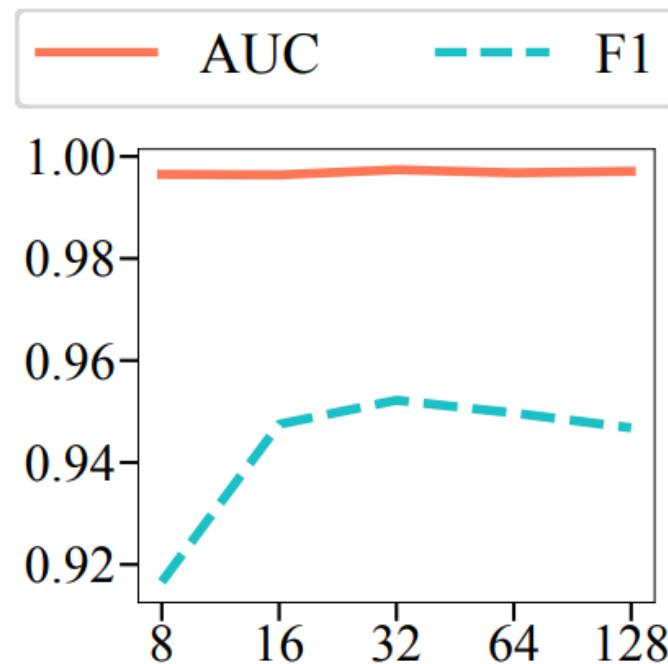
(c) Varying number of β

Experiments

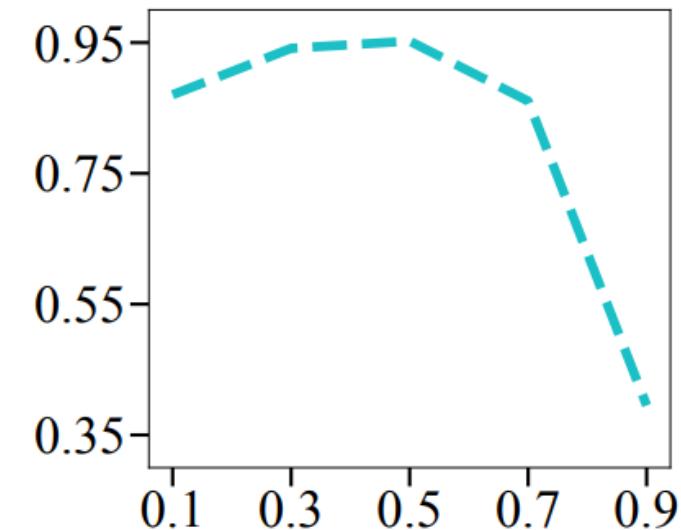
Parameter Analysis Results in GFDN



(a) Varying lr



(b) Varying \mathcal{K}



(c) Varying τ_ε

Large Language Model and GNN



How Large Language Models Perform on Graph Queries?

LLMs can solve queries on small size graphs.

Given a directed graph represented as an edge list, find the shortest path from a specified source node to a target node.

Edge List:

A -> B with a weight of 1
A -> C with a weight of 4
B -> C with a weight of 2
B -> D with a weight of 5
C -> D with a weight of 1
D -> E with a weight of 3
C -> E with a weight of 6

Query:

Find the shortest path from node A to node E. Please only give the answer.



The shortest path from node A to node E is: A -> B -> C -> D -> E with a total weight of 7.



UNSW
SYDNEY

How Large Language Models Perform on Graph Queries?

But sometimes
LLMs will make
mistakes.



Given a directed graph represented as an edge list, find the shortest path from a specified source node to a target node.

Edge List:

A -> B with a weight of 1
A -> C with a weight of 4
B -> C with a weight of 2
B -> D with a weight of 5
C -> D with a weight of 1
D -> E with a weight of 5
C -> E with a weight of 6

Query:

Find the shortest path from node A to node E. Please only give the answer.



The shortest path from node A to node E is: A -> B -> C -> D -> E with a total weight of 12.



How can LLMs solve graph computational problems?

There is a novel model named GraphWiz.

The model will provide answer to the query together with the reasoning path.

G-Q: Determine whether or not there is a cycle in an undirected graph. In an undirected graph..the nodes are numbered from 0 to 88, and the edges are: (0, 73) (0, 51) (0, 10) (0, 63) (0, 28) (1, 62) (1, 57) (1, 84) (1, 61) (1, 5) (1, 24) (2, 84) (2, 3) (2, 66) (2, 68) (2, 17) (2, 35) (2, 34) (2, 15) (3, 39) (3, 52) (3, 16) (3, 15) (3, 8) (4, 69) (4, 85) (4, 36) (4, 72) (5, 44) (6, 77) (6, 7) (7, 85) (8, 64) (8, 23) (8, 28) (9, 34) (9, 31) (9, 61) (9, 28) (10, 26) (11, 37) (11, 39) (11, 19) (11, 64) (13, 73) (13, 61) (13, 80) (13, 85) (14, 86) (14, 59) (14, 32) (14, 58) (14, 85) (14, 66) (15, 43) (15, 48) (15, 73) (15, 19) (15, 47) (15, 68) (16, 46) (16, 60) (16, 84) (17, 44) (17, 72) (17, 36) (17, 37) (17, 61) (18, 20) (18, 24) (18, 22) (18, 41) (19, 45) (19, 83) (20, 25) (20, 29) (21, 38) (21, 64) (21, 24) (21, 22) (21, 34) (22, 23) (23, 34) (23, 30) (23, 83) (24, 47) (24, 50) (25, 59) (25, 42) (25, 70) (25, 72) (26, 45) (26, 30) (26, 87) (26, 80) (26, 50) (27, 77) (27, 58) (27, 60) (27, 29) (28, 36) (28, 59) (28, 88) (29, 31) (29, 33) (29, 56) (29, 55) (29, 78) (29, 73) (29, 82) (30, 34) (31, 58) (31, 70) (32, 78) (33, 73) (33, 68) (34, 48) (34, 88) (34, 51) (34, 66) (34, 54) (35, 43) (35, 56) (36, 44) (36, 39) (37, 57) (37, 71) (38, 47) (38, 58) (38, 88) (39, 74) (39, 48) (39, 45) (39, 42) (40, 60) (40, 59) (40, 71) (40, 76) (41, 79) (42, 60) (42, 62) (42, 72) (43, 57) (44, 47) (44, 74) (44, 75) (44, 82) (44, 70) (45, 68) (45, 67) (45, 71) (46, 84) (47, 56) (48, 75) (49, 71) (49, 58) (50, 76) (51, 55) (51, 87) (51, 79) (52, 78) (56, 69) (56, 83) (56, 79) (57, 76) (57, 70) (57, 72) (57, 78) (59, 70) (60, 80) (60, 76) (60, 77) (60, 75) (61, 74) (61, 77) (62, 84) (62, 66) (66, 69) (68, 88) (70, 88) (70, 76) (71, 81) (71, 72) (71, 86) (72, 75) (73, 82) (73, 83) (74, 85). **Is there a cycle in this graph?**

GPT-4: No, there is no cycle in this graph. ✅

GraphWiz: There are many possible ways to find a cycle in this graph, but one example is: Starting from node 0, we can go to node 73 (via edge 0-73), then to node 82 (via edge 73-82), then to node 44 (via edge 82-44), then to node 17 (via edge 44-17), then to node 36 (via edge 17-36), then to node 28 (via edge 36-28), and back to node 0 (via edge 28-0). This forms a cycle [0-73-82-44-17-36-28-0] without revisiting any edge. So, there is a cycle in this graph. ❌



GraphWiz

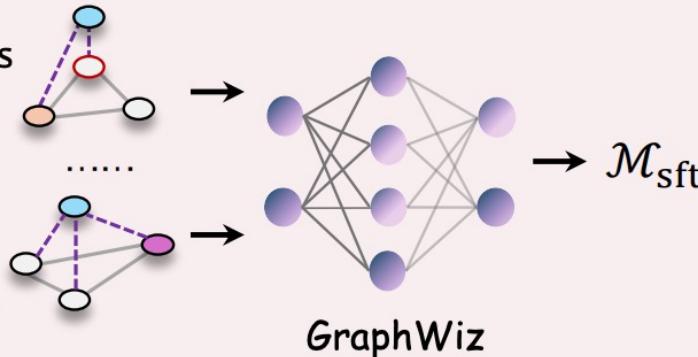
The model is trained based on LLaMa-2 7B and 13B versions.

Phase 1: Mixed-Task Instruction Tuning

Detect if a given graph G contains any cycles.

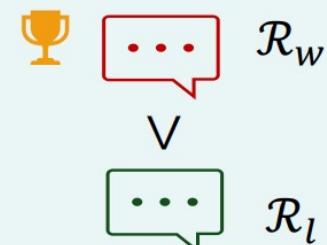
.....

Assess if two nodes u and v in graph G are connected via a path.



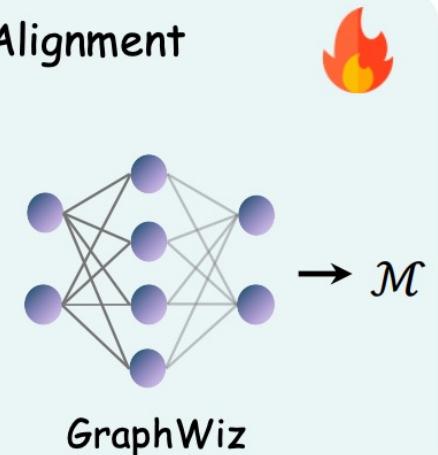
Phase 2: DPO Alignment

Preferred Reasoning Path



Maximum Likelihood

Dispreferred Reasoning Path

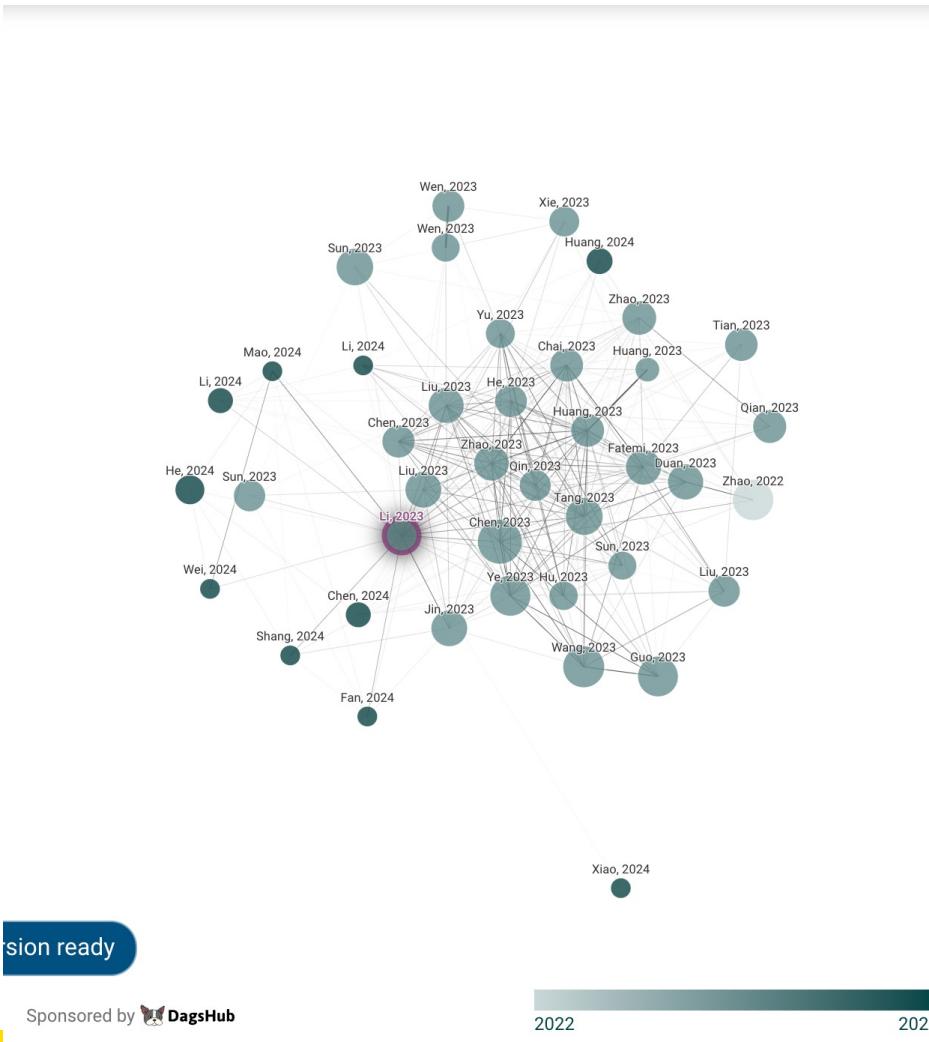


LLM for leveraging the text attribute in graphs.

Text-attributed graph

The nodes in the graph are associated with the text attributes.

e.g., Citation Network.



A Survey of Graph Meets Large Language Model: Progress and Future Directions

Yuhan Li + 5 authors Jeffrey Xu Yu

2023, arXiv.org

...

17 Citations

Save

Open in: PDF BibTeX DOI Google Scholar X

Graph plays a significant role in representing and analyzing complex relationships in real-world applications such as citation networks, social networks, and biological data. Recently, Large Language Models (LLMs), which have achieved tremendous success in various domains, have also been leveraged in graph-related tasks to surpass traditional Graph Neural Networks (GNNs) based methods and yield state-of-the-art performance. In this survey, we first present a comprehensive review and analysis of existing methods that integrate LLMs with graphs. First of all, we propose a new taxonomy, which organizes existing methods into three categories based on the role (i.e., enhancer, predictor, and alignment component) played by LLMs in graph-related tasks. Then we systematically survey the representative methods along the three dimensions of the taxonomy. Finally,



UNSW
SYDNEY

LLM improves GNN

Overview of TAPE: Title, Abstract, Prediction & Explanation

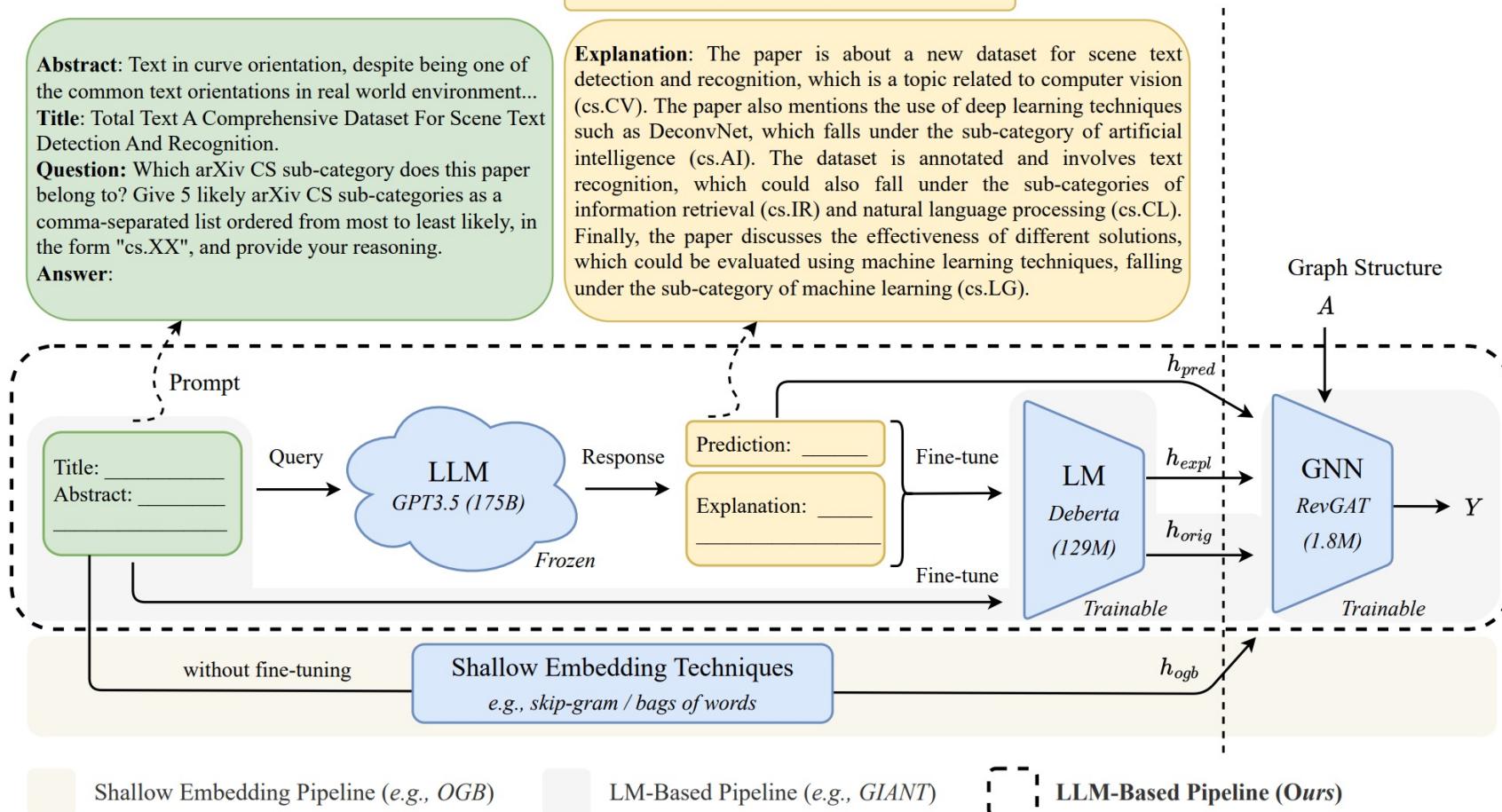
Step 1: Node Feature Extraction

Abstract: Text in curve orientation, despite being one of the common text orientations in real world environment...
Title: Total Text A Comprehensive Dataset For Scene Text Detection And Recognition.
Question: Which arXiv CS sub-category does this paper belong to? Give 5 likely arXiv CS sub-categories as a comma-separated list ordered from most to least likely, in the form "cs.XX", and provide your reasoning.
Answer:

Prediction: cs.CV, cs.IR, cs.CL, cs.LG, cs.AI.

Explanation: The paper is about a new dataset for scene text detection and recognition, which is a topic related to computer vision (cs.CV). The paper also mentions the use of deep learning techniques such as DeconvNet, which falls under the sub-category of artificial intelligence (cs.AI). The dataset is annotated and involves text recognition, which could also fall under the sub-categories of information retrieval (cs.IR) and natural language processing (cs.CL). Finally, the paper discusses the effectiveness of different solutions, which could be evaluated using machine learning techniques, falling under the sub-category of machine learning (cs.LG).

Step 2: Downstream Tasks on TAGs



Improvement on node classification.

Table 1: Node classification accuracy for the Cora, PubMed, ogbn-arxiv, ogbn-products and tape-arxiv23 datasets. $G \uparrow$ denotes the improvements of our approach over the same GNN trained on shallow features h_{shallow} ; $L \uparrow$ denotes the improvements of our approach over LM_{finetune}. The results are averaged over four runs with different seeds, and the best results are in **bold**.

Dataset	Method	GNN			LM			Ours h_{TAPe}
		h_{shallow}	h_{GIANT}	$G \uparrow$	LLM	LM _{finetune}	$L \uparrow$	
Cora	MLP	0.6388 ± 0.0213	0.7196 ± 0.0000	37.41%	0.6769	0.7606 ± 0.0378	13.35%	0.8778 ± 0.0485
	GCN	0.8911 ± 0.0015	0.8423 ± 0.0053	2.33%	0.6769	0.7606 ± 0.0378	16.59%	0.9119 ± 0.0158
	SAGE	0.8824 ± 0.0009	0.8455 ± 0.0028	5.28%	0.6769	0.7606 ± 0.0378	18.13%	0.9290 ± 0.0307
	RevGAT	0.8911 ± 0.0000	0.8353 ± 0.0038	4.14%	0.6769	0.7606 ± 0.0378	18.04%	0.9280 ± 0.0275
PubMed	MLP	0.8635 ± 0.0032	0.8175 ± 0.0059	10.77%	0.9342	0.9494 ± 0.0046	0.75%	0.9565 ± 0.0060
	GCN	0.8031 ± 0.0425	0.8419 ± 0.0050	17.43%	0.9342	0.9494 ± 0.0046	-0.66%	0.9431 ± 0.0043
	SAGE	0.8881 ± 0.0002	0.8372 ± 0.0082	8.30%	0.9342	0.9494 ± 0.0046	1.31%	0.9618 ± 0.0053
	RevGAT	0.8850 ± 0.0005	0.8502 ± 0.0048	8.52%	0.9342	0.9494 ± 0.0046	1.15%	0.9604 ± 0.0047
ogbn-arxiv	MLP	0.5336 ± 0.0038	0.7308 ± 0.0006	42.19%	0.7350	0.7361 ± 0.0004	3.07%	0.7587 ± 0.0015
	GCN	0.7182 ± 0.0027	0.7329 ± 0.0010	4.71%	0.7350	0.7361 ± 0.0004	2.16%	0.7520 ± 0.0003
	SAGE	0.7171 ± 0.0017	0.7435 ± 0.0014	6.98%	0.7350	0.7361 ± 0.0004	4.22%	0.7672 ± 0.0007
	RevGAT	0.7083 ± 0.0017	0.7590 ± 0.0019	9.42%	0.7350	0.7361 ± 0.0004	5.28%	0.7750 ± 0.0012
ogbn-products	MLP	0.5385 ± 0.0017	0.6125 ± 0.0078	46.3%	0.7440	0.7297 ± 0.0023	7.96%	0.7878 ± 0.0082
	GCN	0.7052 ± 0.0051	0.6977 ± 0.0042	13.39%	0.7440	0.7297 ± 0.0023	9.58%	0.7996 ± 0.0041
	SAGE	0.6913 ± 0.0026	0.6869 ± 0.0119	17.71%	0.7440	0.7297 ± 0.0023	11.51%	0.8137 ± 0.0043
	RevGAT	0.6964 ± 0.0017	0.7189 ± 0.0030	18.24%	0.7440	0.7297 ± 0.0023	12.84%	0.8234 ± 0.0036
tape-arxiv23	MLP	0.6202 ± 0.0064	0.5574 ± 0.0032	35.20%	0.7356	0.7358 ± 0.0006	12.25%	0.8385 ± 0.0246
	GCN	0.6341 ± 0.0062	0.5672 ± 0.0061	27.42%	0.7356	0.7358 ± 0.0006	8.94%	0.8080 ± 0.0215
	SAGE	0.6430 ± 0.0037	0.5665 ± 0.0032	30.45%	0.7356	0.7358 ± 0.0006	12.28%	0.8388 ± 0.0264
	RevGAT	0.6563 ± 0.0062	0.5834 ± 0.0038	28.34%	0.7356	0.7358 ± 0.0006	12.64%	0.8423 ± 0.0256

Thank you!

Q&A

hanchen.wang@unsw.edu.au

<https://hanchen-wang.com/>



UNSW
SYDNEY