Question: Is all the information necessary for reasoning on knowledge graphs? 🤔

ICLR

Less is More: One-shot Subgraph Reasoning on Large-scale Knowledge Graphs

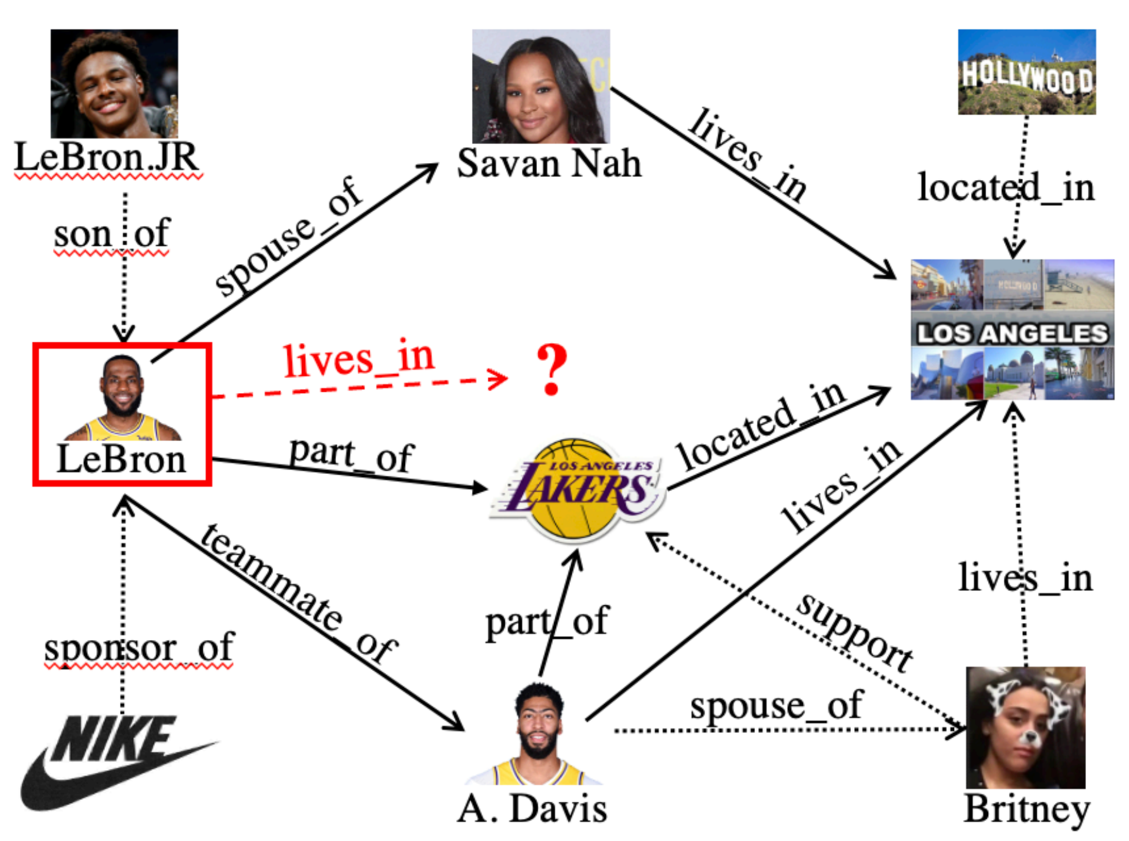
Zhanke Zhou, Yongqi Zhang, Jiangchao Yao, Quanming Yao, Bo Han

TMLR TRUSTWORTHY MACHINE LEARNING AND REASONING
香港浸會大學 HONG KONG BAPTIST UNIVERSITY
清華大学 Tsinghua University
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

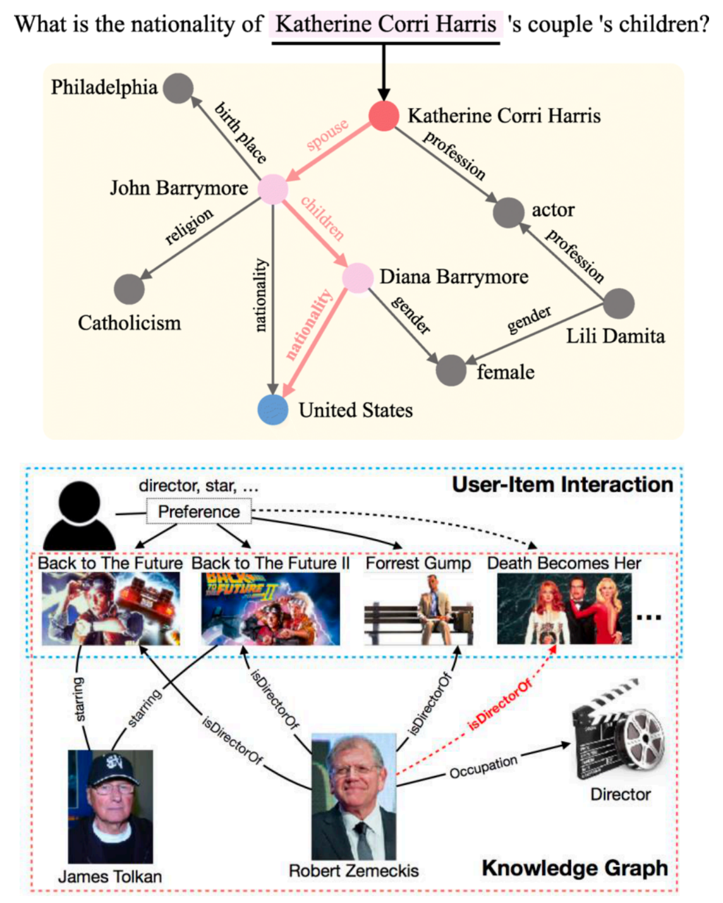
Paper Code Slides

Problem: Link Prediction on KG

KG reasoning with query:
(LeBron, lives_in, ?)

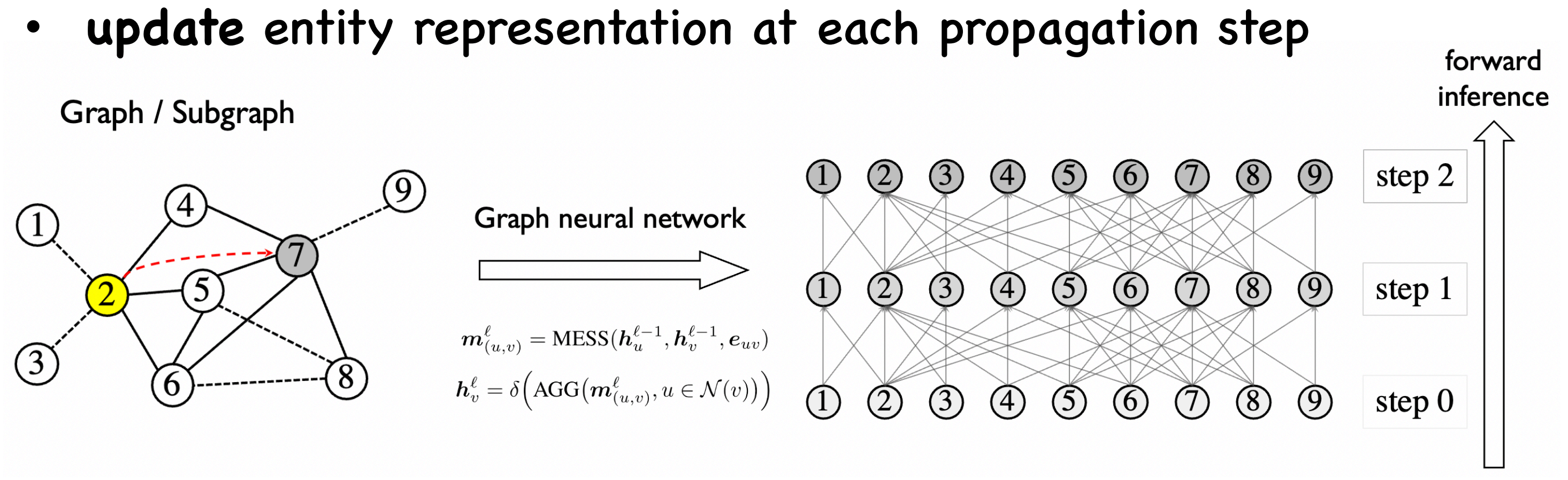


Applications: QA / RecSys

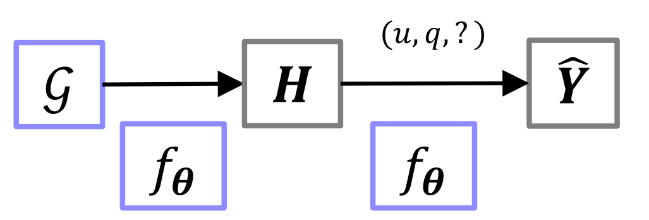


Structural models (e.g., GNNs) for KG reasoning

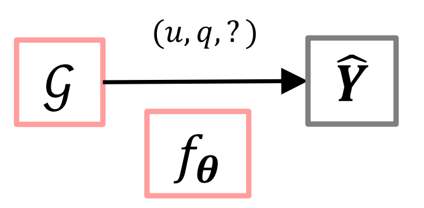
- **propagate** the message with the graph structure
- **update** entity representation at each propagation step



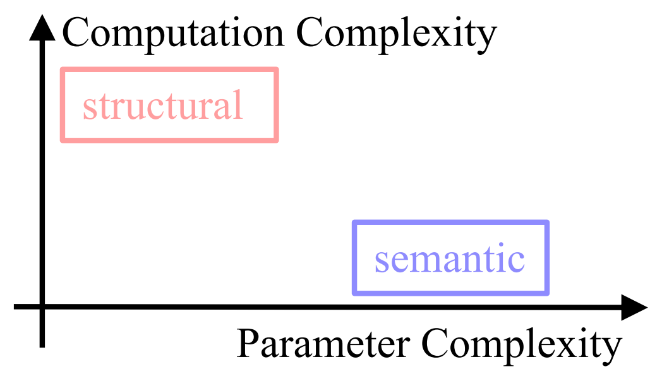
The Scalability Issue



semantic models



structural models



complexity comparison

semantic models (computation-efficient but parameter-expensive)

- $p(u,q,v)$ is measured by a scoring function with representations $\boldsymbol{h}_u, \boldsymbol{h}_q, \boldsymbol{h}_v$

structural models (parameter-efficient but computation-expensive)

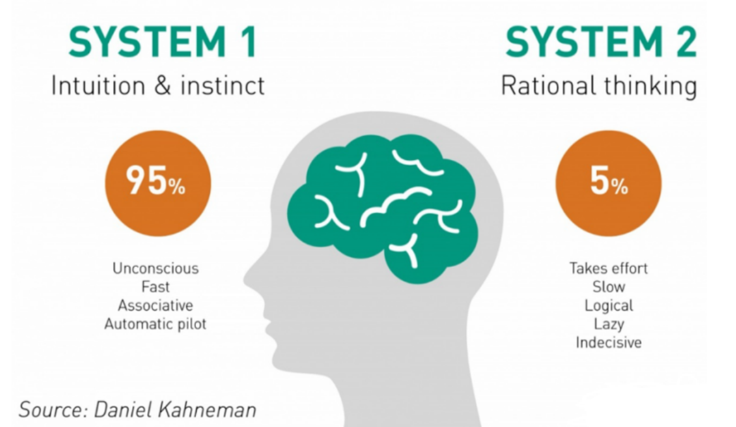
- learn the structures by leveraging the relational paths between u and v
- or use the graph structure for reasoning, capturing more complex semantics

➔ f_θ acts on $\mathcal{G}$ to obtain $\widehat{Y}$ of all entities

➔ The whole graph ($\mathcal{G}$), model (f_θ), and prediction ($\widehat{Y}$) are coupled

How to efficiently and effectively conduct subgraph reasoning on KG? 🤔

Method: one-shot-subgraph reasoning



Source: Daniel Kahneman

Design principle

- first to efficiently identify a subgraph (system1) ➔ sampler
- then effectively reason on the subgraph (system2) ➔ predictor

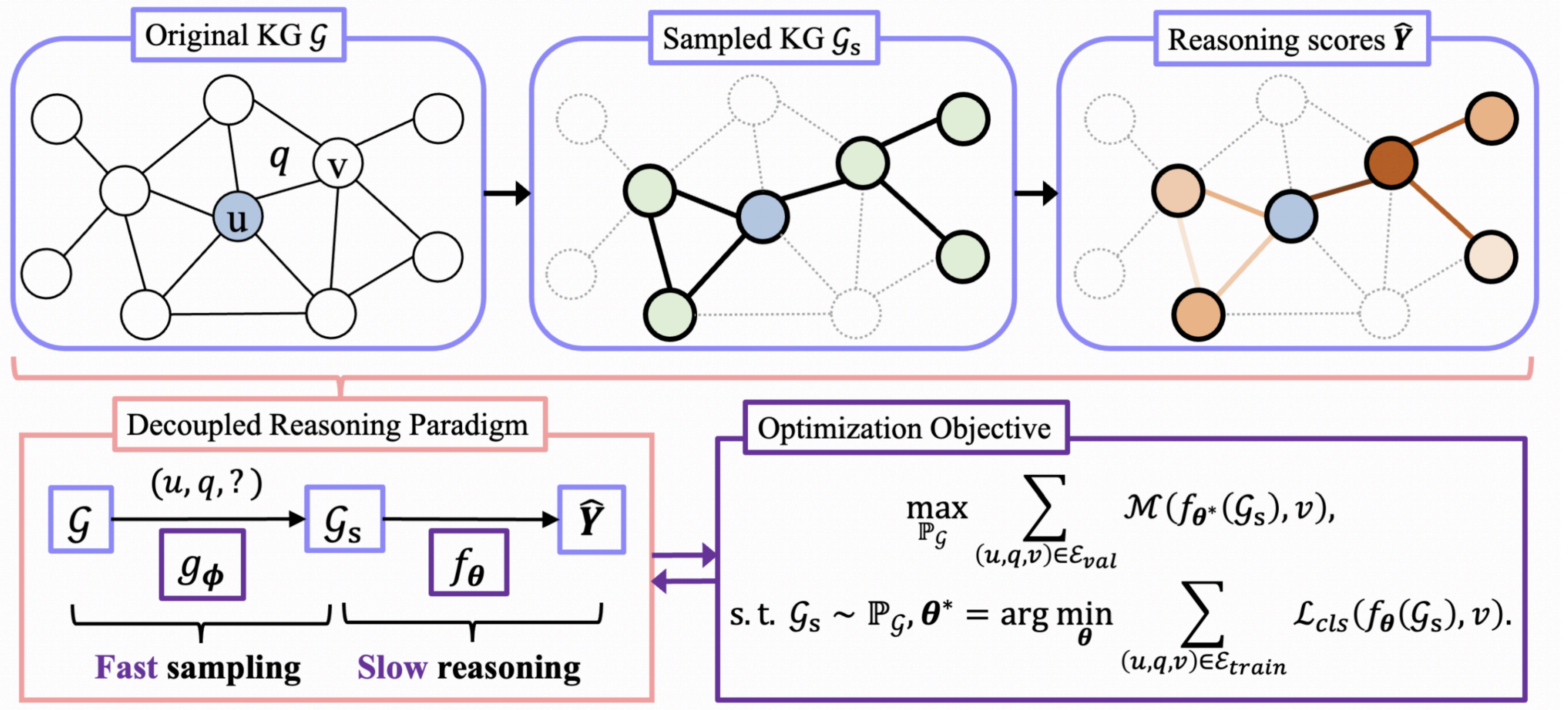
- Only partial knowledge stored in human brain is relevant to a question
- Generating candidates and then ranking the promising ones are common

Definition 1 (One-shot-subgraph Link Prediction on Knowledge Graphs). *Instead of directly predicting on the original graph $\mathcal{G}$, the prediction procedure is decoupled to two-fold: (1) one-shot sampling of a query-dependent subgraph and (2) prediction on this subgraph. The prediction pipeline becomes*

$$\mathcal{G} \xmapsto{g_\phi,(u,q)} \mathcal{G}_s \xmapsto{f_\theta} \hat{\boldsymbol{Y}}, \tag{1}$$

where the sampler g_ϕ generates only one subgraph $\mathcal{G}_s$ (satisfies $|\mathcal{V}_s| \ll |\mathcal{V}|, |\mathcal{E}_s| \ll |\mathcal{E}|$) conditioned on the given query $(u,q,?)$. Based on subgraph $\mathcal{G}_s$, the predictor f_θ outputs the final predictions $\hat{\boldsymbol{Y}}$.

Implementation


Original KG $\mathcal{G}$
q
v
u
Sampled KG $\mathcal{G}_s$
Reasoning scores $\widehat{\boldsymbol{Y}}$
Decoupled Reasoning Paradigm
$\mathcal{G}$
$(u,q,?)$
$\mathcal{G}_s$
$\widehat{\boldsymbol{Y}}$
g_ϕ
f_θ
Fast sampling
Slow reasoning
Optimization Objective

$$\max_{\mathbb{P}_\mathcal{G}} \sum_{(u,q,v)\in\mathcal{E}_{val}} \mathcal{M}(f_{\boldsymbol{\theta}^*}(\mathcal{G}_s), v),$$

$$\text{s.t. } \mathcal{G}_s \sim \mathbb{P}_\mathcal{G}, \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{(u,q,v)\in\mathcal{E}_{train}} \mathcal{L}_{cls}(f_{\boldsymbol{\theta}}(\mathcal{G}_s), v).$$

The three key steps of one-shot-subgraph reasoning are

1. generate the sampling distribution

Non-parametric indicator: $\boldsymbol{p}^{(k+1)} \leftarrow \alpha \cdot \boldsymbol{s} + (1-\alpha) \cdot \boldsymbol{D}^{-1}\boldsymbol{A} \cdot \boldsymbol{p}^{(k)}$,

2. extract a subgraph with top entities and edges

Entity Sampling: $\mathcal{V}_s \leftarrow \text{TopK}\Big(\mathcal{V},\, \boldsymbol{p},\, K{=}r^q_\mathcal{V}{\times}|\mathcal{V}|\Big)$,

Edge Sampling: $\mathcal{E}_s \leftarrow \text{TopK}\Big(\mathcal{E},\, \{\boldsymbol{p}_x{\cdot}\boldsymbol{p}_o : x,o{\in}\mathcal{V}_s, (x,r,o){\in}\mathcal{E}\},\, K{=}r^q_\mathcal{E}{\times}|\mathcal{E}|\Big)$.

3. inference on the subgraph and get the final prediction

Indicating: $\boldsymbol{h}^0_o \leftarrow \mathbb{1}(o=u)$,

Propagation: $\boldsymbol{h}^{l+1}_o \leftarrow \texttt{DROPOUT}\Big(\texttt{ACT}\Big(\texttt{AGG}\big\{\texttt{MESS}(\boldsymbol{h}^l_x, \boldsymbol{h}^l_r, \boldsymbol{h}^l_o) : (x,r,o) \in \mathcal{E}_s\big\}\Big)\Big)$

Experiments

Table 1: Empirical results of WN18RR, NELL-995, YAGO3-10 datasets. Best performance is indicated by the **bold face** numbers, and the underline means the second best. "–" means unavailable results. "H@1" and "H@10" are short for Hit@1 and Hit@10 (in percentage), respectively.

type	models	WN18RR			NELL-995			YAGO3-10		
		MRR↑	H@1↑	H@10↑	MRR↑	H@1↑	H@10↑	MRR↑	H@1↑	H@10↑
Semantic Models	ConvE	0.427	39.2	49.8	0.511	44.6	61.9	0.520	45.0	66.0
	QuatE	0.480	44.0	55.1	0.533	46.6	64.3	0.379	30.1	53.4
	RotatE	0.477	42.8	57.1	0.508	44.8	60.8	0.495	40.2	67.0
Structural Models	MINERVA	0.448	41.3	51.3	0.513	41.3	63.7	–	–	–
	DRUM	0.486	42.5	58.6	0.532	46.0	**66.2**	0.531	45.3	67.6
	RNNLogic	0.483	44.6	55.8	0.416	36.3	47.8	0.554	<u>50.9</u>	62.2
	CompGCN	0.479	44.3	54.6	0.463	38.3	59.6	0.489	39.5	58.2
	DPMPN	0.482	44.4	55.8	0.513	45.2	61.5	0.553	48.4	67.9
	NBFNet	<u>0.551</u>	<u>49.7</u>	**66.6**	0.525	45.1	63.9	0.550	47.9	68.3
	RED-GNN	0.533	48.5	<u>62.4</u>	<u>0.543</u>	<u>47.6</u>	<u>65.1</u>	<u>0.559</u>	48.3	<u>68.9</u>
	one-shot-subgraph	**0.567**	**51.4**	**66.6**	**0.547**	**48.5**	<u>65.1</u>	**0.606**	**54.0**	**72.1**

Table 2: Empirical results of two OGB datasets (Hu et al., 2020) with regard to official leaderboards.

type	models	OGBL-BIOKG			OGBL-WIKIKG2		
		Test MRR↑	Valid MRR↑	#Params↓	Test MRR↑	Valid MRR↑	#Params↓
Semantic Models	TripleRE	0.8348	0.8360	469,630,002	0.5794	0.6045	500,763,337
	AutoSF	0.8309	0.8317	93,824,000	0.5458	0.5510	500,227,800
	PairRE	0.8164	0.8172	187,750,000	0.5208	0.5423	500,334,800
	ComplEx	0.8095	0.8105	187,648,000	0.4027	0.3759	1,250,569,500
	DistMult	0.8043	0.8055	187,648,000	0.3729	0.3506	1,250,569,500
	RotatE	0.7989	0.7997	187,597,000	0.4332	0.4353	1,250,435,750
	TransE	0.7452	0.7456	187,648,000	0.4256	0.4272	1,250,569,500
Structural Models	**one-shot-subgraph**	**0.8430**	**0.8435**	**976,801**	**0.6755**	**0.7080**	**6,831,201**

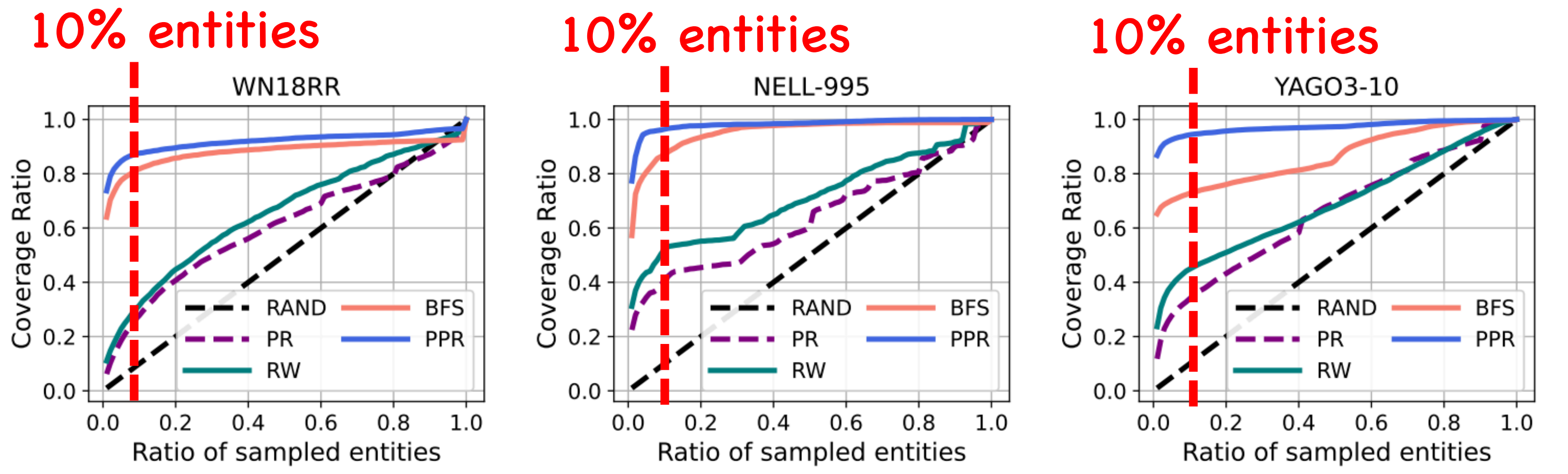


Table 3: Coverage Ratio of different heuristics. **Bold face** numbers indicate the best results in column.

heuristics	WN18RR			NELL-995			YAGO3-10		
	$r^q_\mathcal{V}=0.1$	$r^q_\mathcal{V}=0.2$	$r^q_\mathcal{V}=0.5$	$r^q_\mathcal{V}=0.1$	$r^q_\mathcal{V}=0.2$	$r^q_\mathcal{V}=0.5$	$r^q_\mathcal{V}=0.1$	$r^q_\mathcal{V}=0.2$	$r^q_\mathcal{V}=0.5$
Random Sampling (RAND)	0.100	0.200	0.500	0.100	0.200	0.500	0.100	0.200	0.500
PageRank (PR)	0.278	0.407	0.633	0.405	0.454	0.603	0.340	0.432	0.694
Random Walk (RW)	0.315	0.447	0.694	0.522	0.552	0.710	0.449	0.510	0.681
Breadth-first-searching (BFS)	0.818	0.858	0.898	0.872	0.935	0.982	0.728	0.760	0.848
Personalized PageRank (PPR)	**0.876**	**0.896**	**0.929**	**0.965**	**0.977**	**0.987**	**0.943**	**0.957**	**0.973**

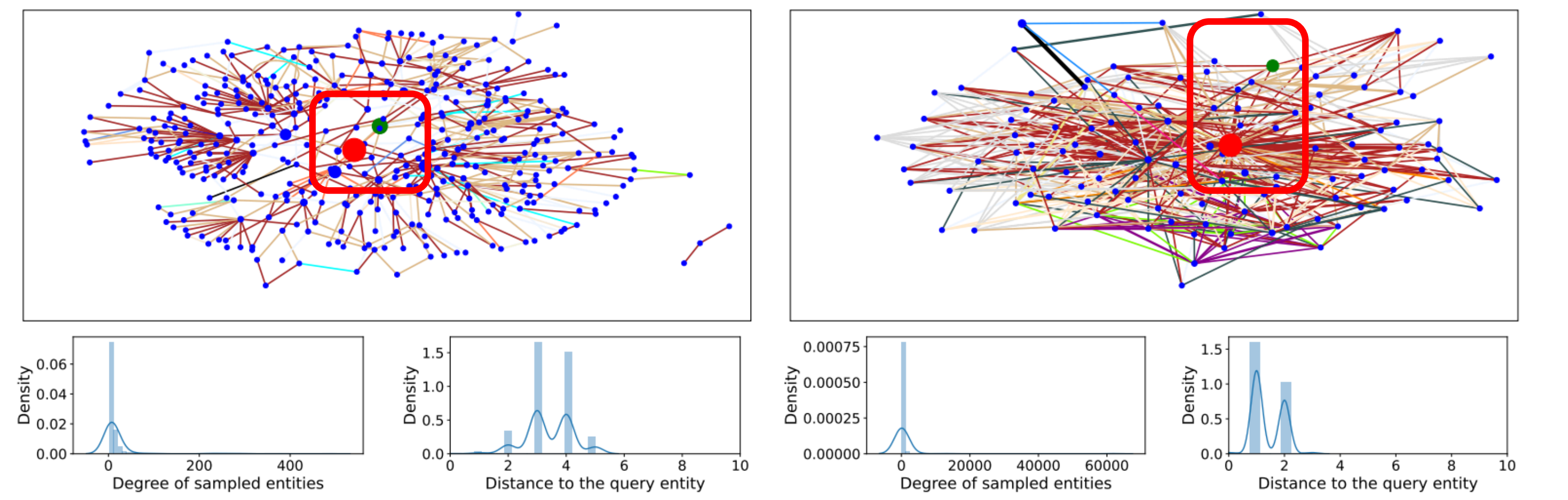


Figure 5: Exemplar subgraphs sampled from WN18RR (left) and YAGO3-10 (right). The red and green nodes indicate the query entity and answer entity. The colors of the edges indicate relation types. The bottom distributions of degree and distance show the statistical properties of each subgraph.

Question: Can we model distant interactions with a single jump in molecular graphs? 🤔

ICLR

Neural Atoms: Propagating Long-Range Interaction In Molecular Graphs hrough Efficient Communication Channel

Xuan Li*, Zhanke Zhou*, Jiangchao Yao, Yu Rong, Lu Zhang, Bo Han

TMLR TRUSTWORTHY MACHINE LEARNING AND REASONING
香港浸會大學 HONG KONG BAPTIST UNIVERSITY
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY
Tencent AI Lab

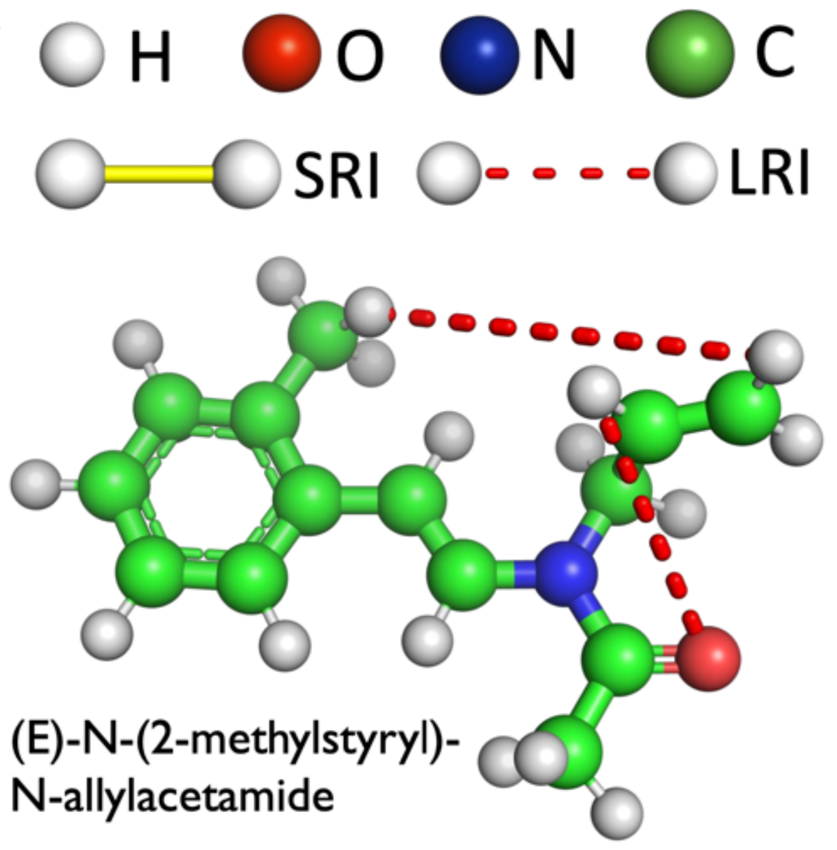
Paper Code

Background: Long-range Interactions

Molecular graphs consist of different types of atom interaction with different properties and functions.

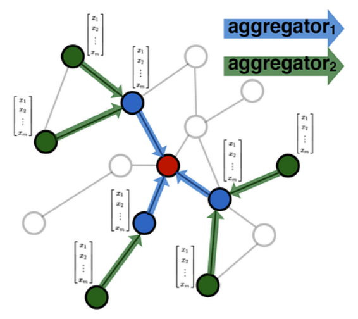
- The short-range interaction (SRI) forms the structure of the molecular graph.

- The long-range interaction (LRI) could determine physical and chemical properties.



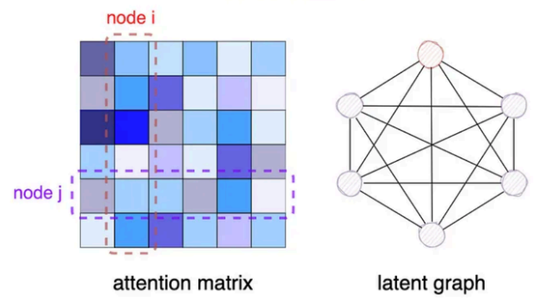
Limitations of existing approaches

Graph Neural Network



Stacking multiple GNN layers to capture LRI?

- **Over-smoothing:** representations become indistinguishable;
- **Over-squashing:** overwhelming information be squashed.



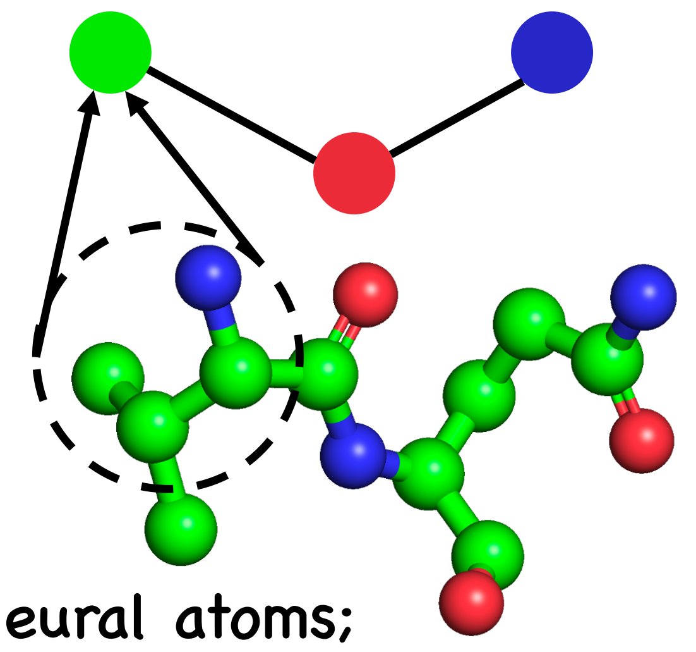
Graph Transformer

Using a **fully connected graph** to capture LRI?

- **Irrelevant interactions:** the LRIs are naturally sparse;
- **Additional computation:** unnecessary node-pair attentions.

Motivation

Can we model distant interactions with a single jump in molecular graphs?

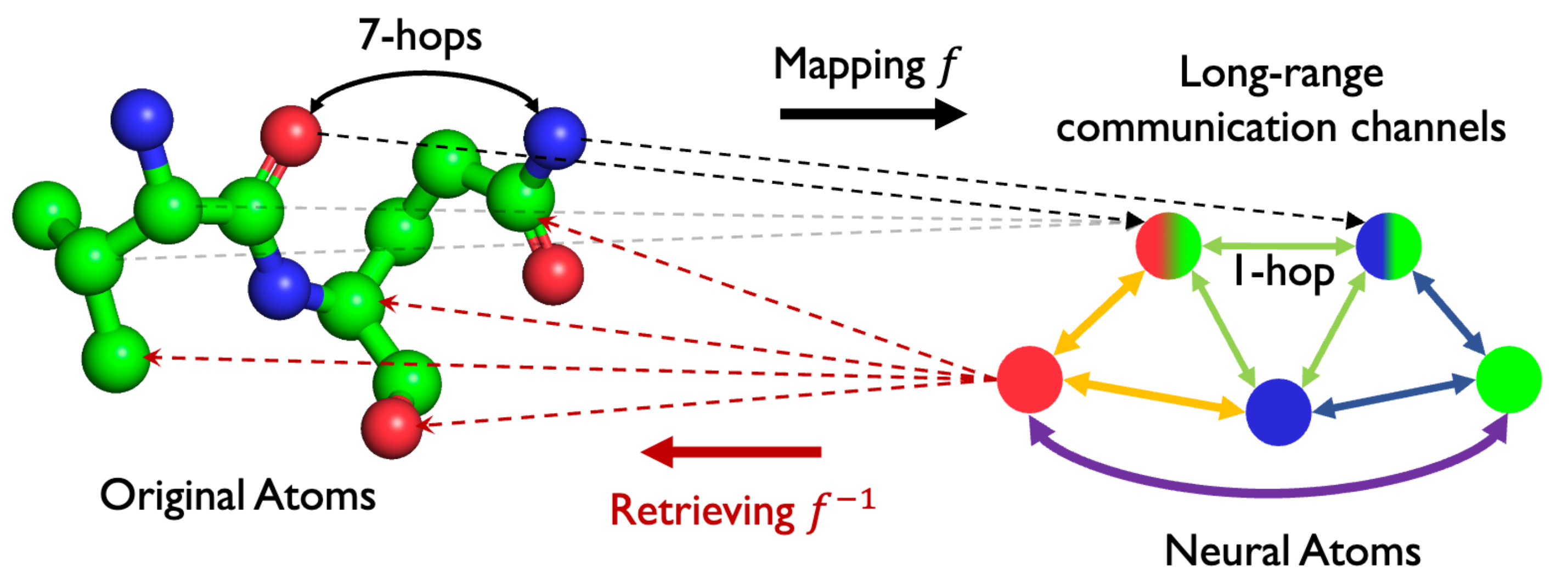


Advantages

- **Learnable projection** from atoms to neural atoms;
- **Reducing** the multi-hop long-range interaction to single-hop;
- GNN-agnostic and **plug-in-and-play**.

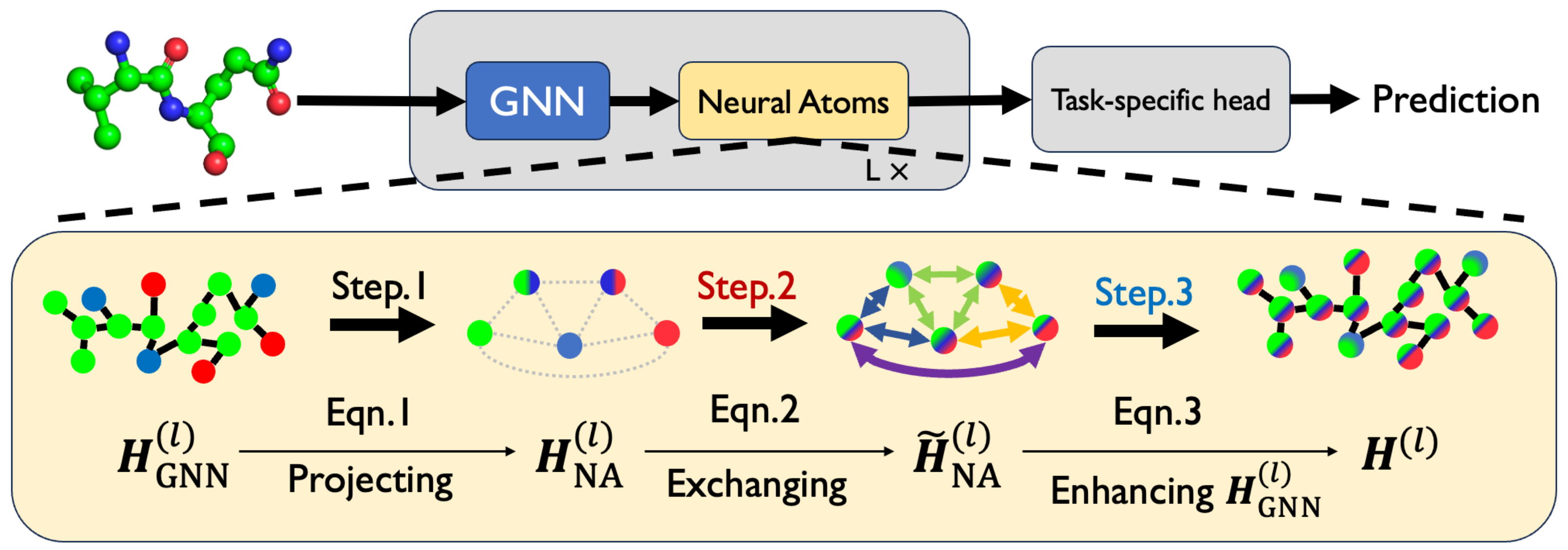
Neural Atoms

Definition 1. *Neural atoms encompass a collection of virtual, parameterized atoms that symbolize a cluster of atoms within a designated molecular graph. The process entails the acquisition of knowledge that enables the transformation of conventional atoms into neural atoms, along with their interactions. This transformation can be technically executed with model-agnostic methodologies.*



Learning to **project** all the original atoms into a few **neural atoms** that **abstract** the collective information of atomic groups in a molecule.

Implementation



Step-1. Project atom representations $H^{(\ell)}_{GNN}$ to neural atom representations $H^{(\ell)}_{\text{NA}}$.

$$\boldsymbol{H}^{(\ell)}_{\text{NA}} = \text{LayerNorm}\left(\boldsymbol{Q}^{(\ell)}_{\text{NA}} \oplus \text{MultiHead}(\boldsymbol{Q}^{(\ell)}_{\text{NA}}, \boldsymbol{H}^{(\ell)}_{\text{GNN}}, \boldsymbol{H}^{(\ell)}_{\text{GNN}})\right)$$

Step-2. Exchange information among neural atoms $H^{(\ell)}_{\text{NA}} \mapsto \tilde{H}^{(\ell)}_{\text{NA}}$.

$$\tilde{\boldsymbol{H}}^{(\ell)}_{\text{NA}} = \text{LayerNorm}\left(\boldsymbol{H}^{(\ell)}_{\text{NA}} \oplus \text{MultiHead}(\boldsymbol{H}^{(\ell)}_{\text{NA}}, \boldsymbol{H}^{(\ell)}_{\text{NA}}, \boldsymbol{H}^{(\ell)}_{\text{NA}})\right)$$

Step-3. Project neural atoms back and **enhance** the atoms' representation $(H^{(\ell)}_{GNN}, \tilde{H}^{(\ell)}_{\text{NA}}) \mapsto H^{(\ell)}$.

$$\boldsymbol{H}^{(\ell)} = \boldsymbol{H}^{(\ell)}_{\text{GNN}} \oplus \tilde{\boldsymbol{A}}^{(\ell)}_{\text{NA}} \tilde{\boldsymbol{H}}^{(\ell)}_{\text{NA}}, \quad \text{s.t.} \quad \tilde{\boldsymbol{A}}^{(\ell)}_{\text{NA}} = \text{Aggregate}\left(\{\hat{\boldsymbol{A}}_m\}_{m=1}^{M}\right)^{\top} \in \mathbb{R}^{N \times K}$$

Empirical Study

→ Neural atoms can **boost** the performance of various GNNs up to **27.32%**.

Table 1: Test performance on three LRGB datasets. Shown is the mean ± s.d. of 4 runs.

Model	Peptides-func	Peptides-struct	PCQM-Contact
	AP ↑	MAE ↓	MRR ↑
Transformer+LapPE	0.6326 ± 0.0126	0.2529 ± 0.0016	0.3174 ± 0.0020
SAN+LapPE	0.6384 ± 0.0121	0.2683 ± 0.0043	0.3350 ± 0.0003
GraphGPS	0.6535 ± 0.0041	0.2500 ± 0.0005	0.3337 ± 0.0006
GCN	0.5930 ± 0.0023	0.3496 ± 0.0013	0.2329 ± 0.0009
+ Neural Atoms	**0.6220 ± 0.0046**	**0.2606 ± 0.0027**	**0.2534 ± 0.0200**
GINE	0.5498 ± 0.0079	0.3547 ± 0.0045	0.3180 ± 0.0027
+ Neural Atoms	**0.6154 ± 0.0157**	**0.2553 ± 0.0005**	**0.3126 ± 0.0021**
GCNII	0.5543 ± 0.0078	0.3471 ± 0.0010	0.3161 ± 0.0004
+ Neural Atoms	**0.5996 ± 0.0033**	**0.2563 ± 0.0020**	**0.3049 ± 0.0006**
GatedGCN	0.5864 ± 0.0077	0.3420 ± 0.0013	0.3218 ± 0.0011
+ Neural Atoms	**0.6562 ± 0.0075**	**0.2585 ± 0.0017**	**0.3258 ± 0.0003**
GatedGCN+RWSE	0.6069 ± 0.0035	0.3357 ± 0.0006	0.3242 ± 0.0008
+ Neural Atoms	**0.6591 ± 0.0050**	**0.2568 ± 0.0005**	**0.3262 ± 0.0010**

→ Neural atoms **without 3D information** and **half the #params.** achieve competitiveness or outperform the Ewald-based approach (previous SOTA).

Table 2: Validation energy MAE and MSE comparison on OE62 dataset.

	Energy MAE ↓	Energy MSE ↓	Number of Params.
SchNet (Schütt et al., 2017)	0.1351	0.0658	2.75 M
+ Ewald Block	**0.0811**	**0.0301**	12.21 M
+ Neural Atoms	**0.0834**	**0.0309**	2.63 M
PaiNN (Schütt et al., 2021)	0.6049	0.0133	12.52 M
+ Ewald Block	**0.0590**	**0.0134**	15.68 M
+ Neural Atoms	**0.0558**	**0.0122**	6.05 M
DimeNet++ (Gasteiger et al., 2020)	0.0501	0.0117	2.76 M
+ Ewald Block	**0.0479**	**0.0107**	4.75 M
+ Neural Atoms	0.0551	0.0129	1.97 M

→ Neural Atoms are **better** than multiple fully connected Virtual Nodes.

Table 14: Performance for virtual nodes (VNs) and neural atoms (NAs) in Peptide-Func, evaluated by AP (the higher, the better).

Model	Method	#VNs /#NAs = 5	#VNs /#NAs = 15	#VNs /#NAs = 75	#VNs /#NAs = 135
GCN	VNs	0.5566	0.5543	0.5568	0.5588
	NAs	**0.5962**	**0.5859**	**0.5903**	**0.6220**
GINE	VNs	0.5437	0.5500	0.5426	0.5426
	NAs	**0.6107**	**0.6128**	**0.6147**	**0.6154**
GCNII	VNs	0.5086	0.5106	0.5077	0.5083
	NAs	**0.6061**	**0.5862**	**0.5909**	**0.5996**
GatedGCN	VNs	0.5810	0.5868	0.5761	0.5810
	NAs	**0.6660**	**0.6533**	**0.6562**	**0.6562**

→ Neural Atoms have **sparse** attention and establish **inter-communications**.

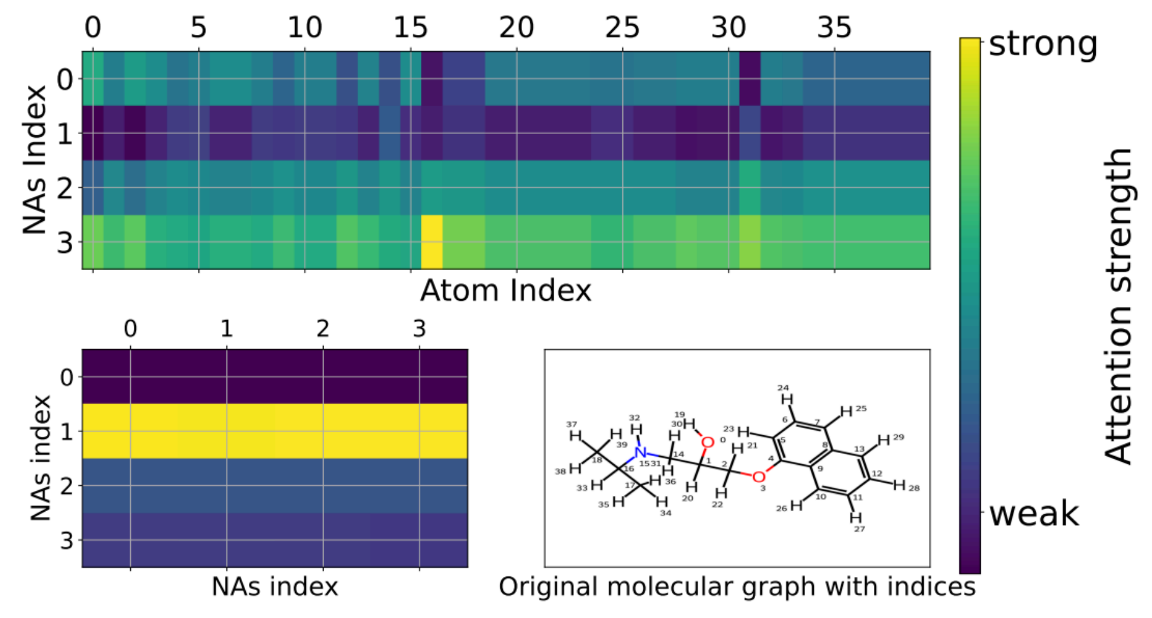


Figure 17: Mutagenicity test set index-18

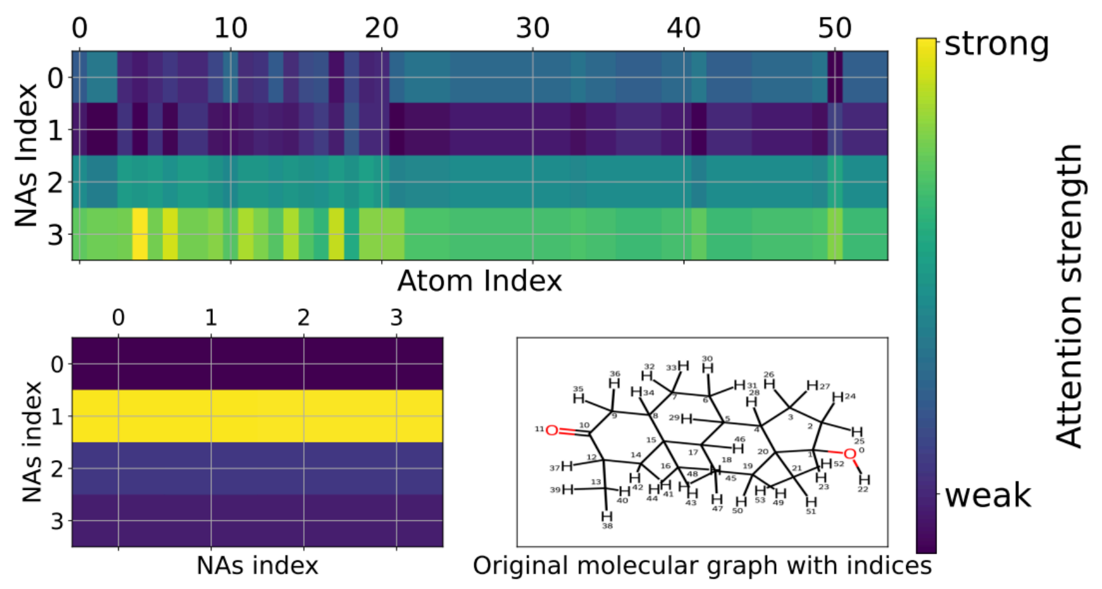


Figure 19: Mutagenicity test set index-29

Paper Code

AdaProp: Learning Adaptive Propagation for Graph Neural Network based Knowledge Graph Reasoning

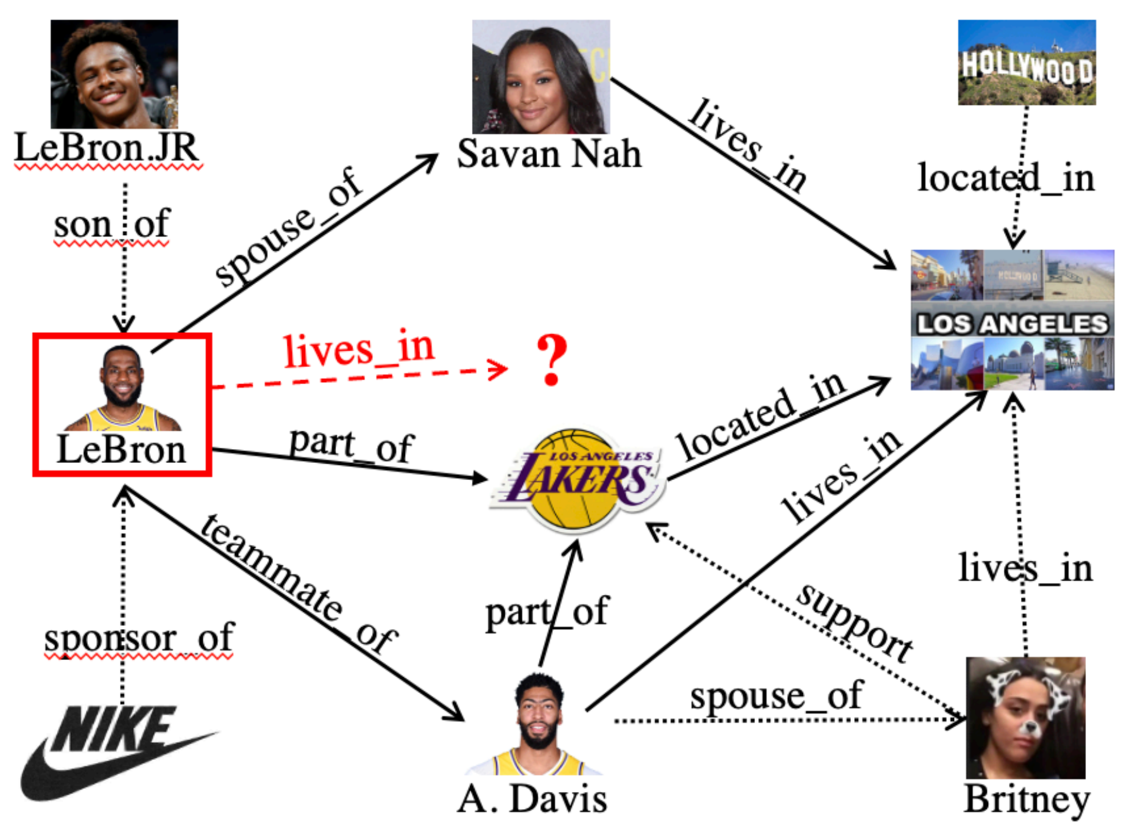
*Yongqi Zhang, *Zhanke Zhou, Quanming Yao, Xiaowen Chu, Bo Han

Contact: zhangyongqi@4paradigm.com, cszkzhou@comp.hkbu.edu.hk

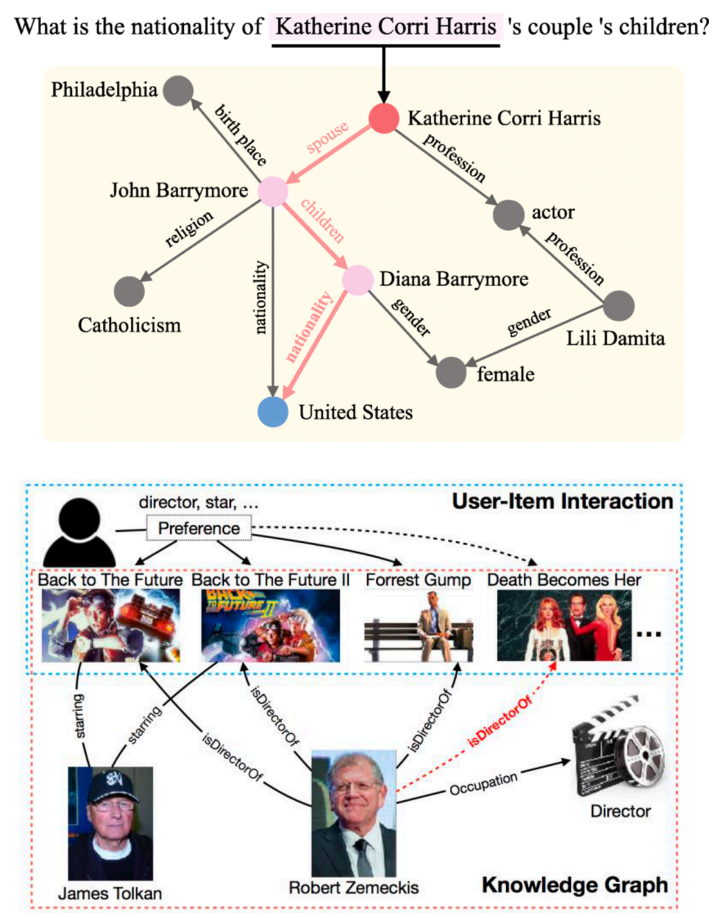
TL;DR: An important design component of GNN-based KG reasoning methods is called the propagation path, which contains a set of involved entities in each propagation step. Existing methods use hand-designed propagation paths, ignoring the correlation between the entities and the query relation. In addition, the number of involved entities will explosively grow at larger propagation steps. In this work, we are motivated to learn an adaptive propagation path in order to filter out irrelevant entities while preserving promising targets.

Background: KG Reasoning

KG reasoning with query: (LeBron, lives_in, ?)

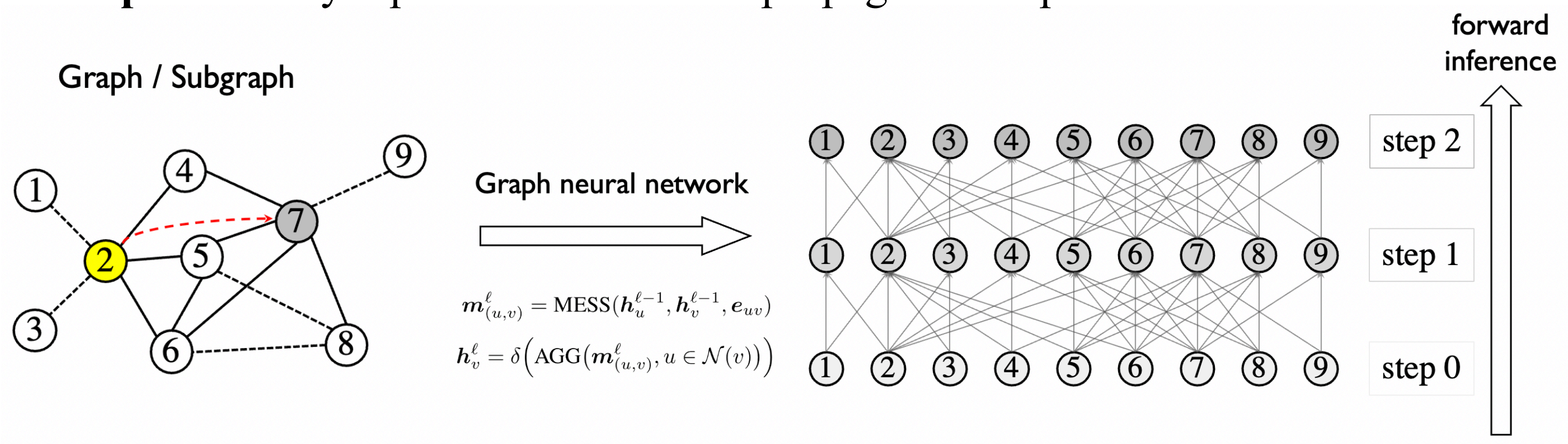


Applications: QA / Recommendation



Graph Neural Network-based methods for KG reasoning

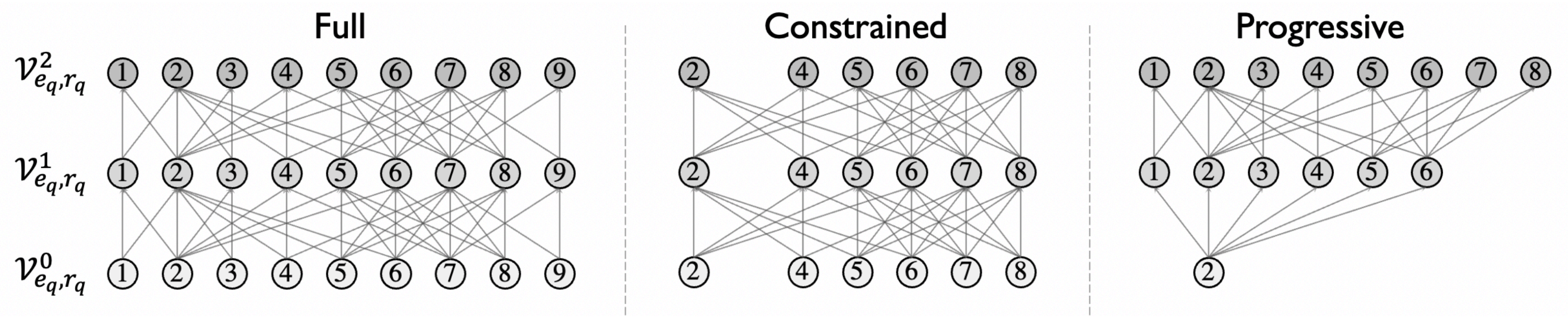
- ☐ **propagate** the message with the graph structure
- ☐ **update** entity representation at each propagation step


Graph / Subgraph
Graph neural network
$m^{\ell}_{(u,v)} = \text{MESS}(h^{\ell-1}_u, h^{\ell-1}_v, e_{uv})$
$h^{\ell}_v = \delta\big(\text{AGG}(m^{\ell}_{(u,v)}, u \in \mathcal{N}(v))\big)$
step 2
step 1
step 0
forward inference

The Propagation Path

Query-dependent propagation path $\widehat{\mathcal{G}}^{L}_{e_q,r_q}$

- ☐ $\hat{\mathcal{G}}^{L}_{e_q,r_q} = \{\mathcal{V}^0_{e_q,r_q}, \mathcal{V}^1_{e_q,r_q}, \dots, \mathcal{V}^L_{e_q,r_q}\}$ as the sets of involved entities
- ☐ in each propagation step for query $(e_q, r_q, ?)$


Full
Constrained
Progressive
$\mathcal{V}^2_{e_q,r_q}$
$\mathcal{V}^1_{e_q,r_q}$
$\mathcal{V}^0_{e_q,r_q}$

Problems when L is large

- ☐ *Full* propagation: large memory cost & over-smoothing
- ☐ *Constrained* propagation: extremely high inference cost
- ☐ *Progressive* propagation: exponentially increased nodes

Problem & Challenges

Problem formulation: Reduce the size of propagation path through **sampling**

$$\widehat{\mathcal{G}}^{L}_{e_q,r_q} = \{\mathcal{V}^0_{e_q,r_q}, \mathcal{V}^1_{e_q,r_q}, \dots, \mathcal{V}^L_{e_q,r_q}\},$$

$$\text{s.t. } \mathcal{V}^{\ell}_{e_q,r_q} = \begin{cases} \{e_q\} & \ell = 0 \\ S(\mathcal{V}^{\ell-1}_{e_q,r_q}) & \ell = 1 \dots L \end{cases}.$$

Two challenges of the sampling strategy $S(\cdot)$

- ☐ the target answer e_a is unknown given $(e_q, r_q, ?)$
- ☐ semantic dependency is complex

Existing sampling approaches are not applicable

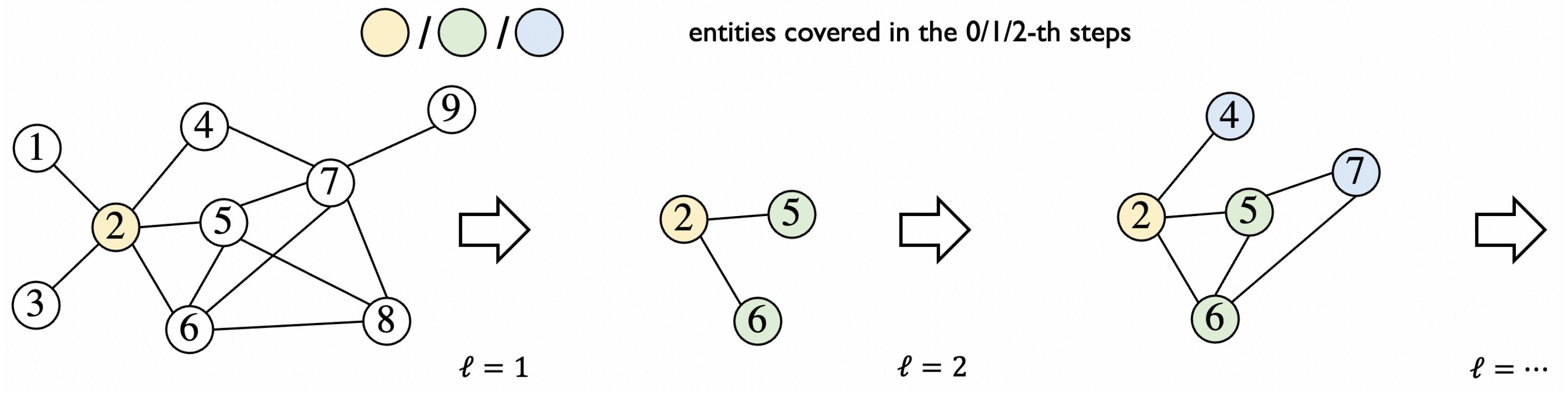
- ☐ no target preserving
- ☐ no relation consideration
- ☐ no direct supervision

Method: adaptively sample semantically relevant entities during propagation

Design1: Connection-preserving Incremental Sampling

☐ **Key idea:** Preserve the previous entities & sample from the newly visited ones $\mathcal{V}^0_{e_q,r_q} \subseteq \mathcal{V}^1_{e_q,r_q} \cdots \subseteq \mathcal{V}^L_{e_q,r_q}$

☐ **Incremental sampling with only linear complexity**



☐ **Details in each step: Candidate generation and sampling**

Candidate generation:
the newly-visit neighboring entities of last step

$$\overline{\mathcal{V}}^{\ell}_{e_q,r_q} := \text{CAND}(\mathcal{V}^{\ell-1}_{e_q,r_q}) = \mathcal{N}(\mathcal{V}^{\ell-1}_{e_q,r_q}) \setminus \mathcal{V}^{\ell-1}_{e_q,r_q}.$$

e.g. ① ③ ④ ⑤ ⑥ when $l = 1$
① ③ ④ ⑦ ⑧ when $l = 2$

Candidate sampling:
sample K entities without replacement from candidates

$$\mathcal{V}^{\ell}_{e_q,r_q} := \mathcal{V}^{\ell-1}_{e_q,r_q} \cup \text{SAMP}(\overline{\mathcal{V}}^{\ell}_{e_q,r_q}).$$

e.g. ⑤ ⑥ when $l = 1$
④ ⑦ when $l = 2$

Design2: Learning-based and Sematic-aware Distribution

☐ **Key idea:** Introduce a parameterized distribution & borrow knowledge from the GNN $\mathcal{V}^{\ell}_{e_q,r_q} = S(\mathcal{V}^{\ell-1}_{e_q,r_q}; \boldsymbol{\theta}^{\ell})$

Parameterized sampling distribution:

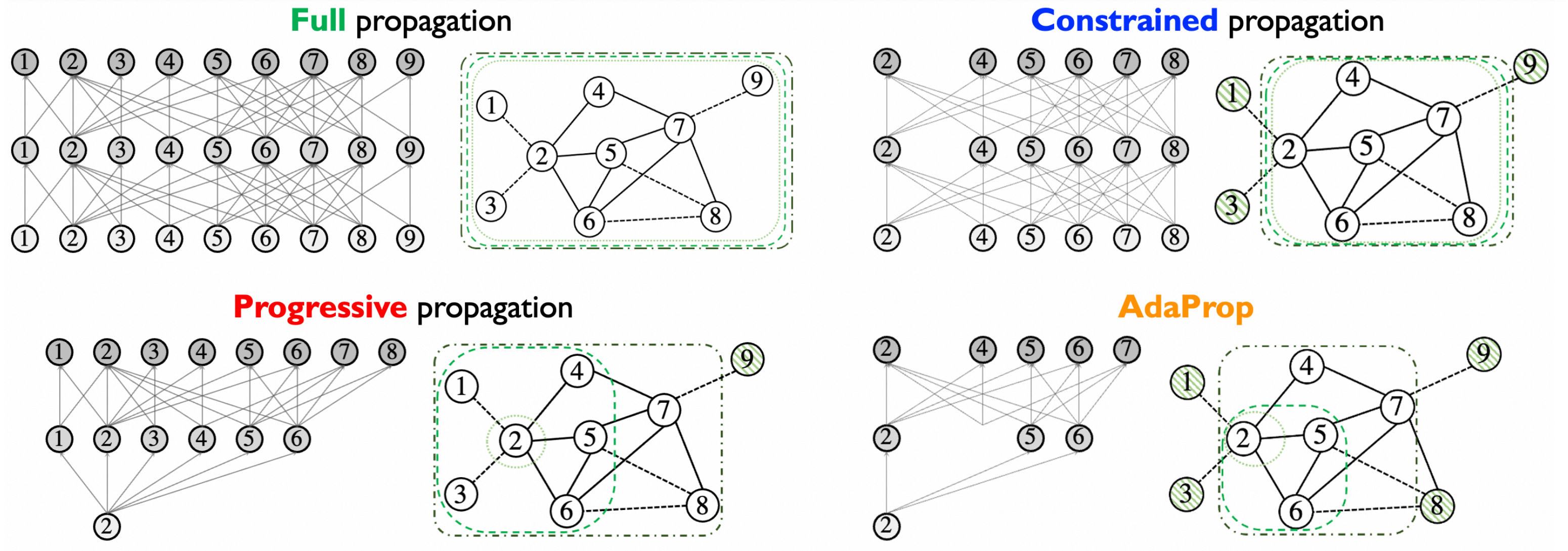
- ☐ Sharing the knowledge in GNN representations $\boldsymbol{h}^{\ell}_e$
- ☐ Adaptive based on the learnable parameters $\boldsymbol{\theta}^{\ell}$

$$p^{\ell}(e) := \exp\left(g(\boldsymbol{h}^{\ell}_e; \boldsymbol{\theta}^{\ell})/\tau\right) \Big/ \sum_{e' \in \overline{\mathcal{V}}^{\ell}_{e_q,r_q}} \exp\left(g(\boldsymbol{h}^{\ell}_{e'}; \boldsymbol{\theta}^{\ell})/\tau\right)$$

Learning strategy:

- ☐ Gumbel-trick to enable backward propagation on hard samples.
- ☐ Sampling: get top-K based on gumbel-logits
  $G_e := g(\boldsymbol{h}^{\ell}_e; \boldsymbol{\theta}^{\ell}) - \log(-\log U_e)$ with $U_e \sim \text{Uniform}(0,1)$ for the candidate entities
- ☐ Enable backpropagation: straight-through estimation
  $\boldsymbol{h}^{\ell}_e = \big(1 - \text{no\_grad}(p^{\ell}(e)) + p^{\ell}(e)\big) \cdot \boldsymbol{h}^{\ell}_e$ for the selected entities

An overall comparison with existing propagation schemes



Adaprop achieves smaller complexity, deeper steps and query-dependent.

Comprehensive Experiments

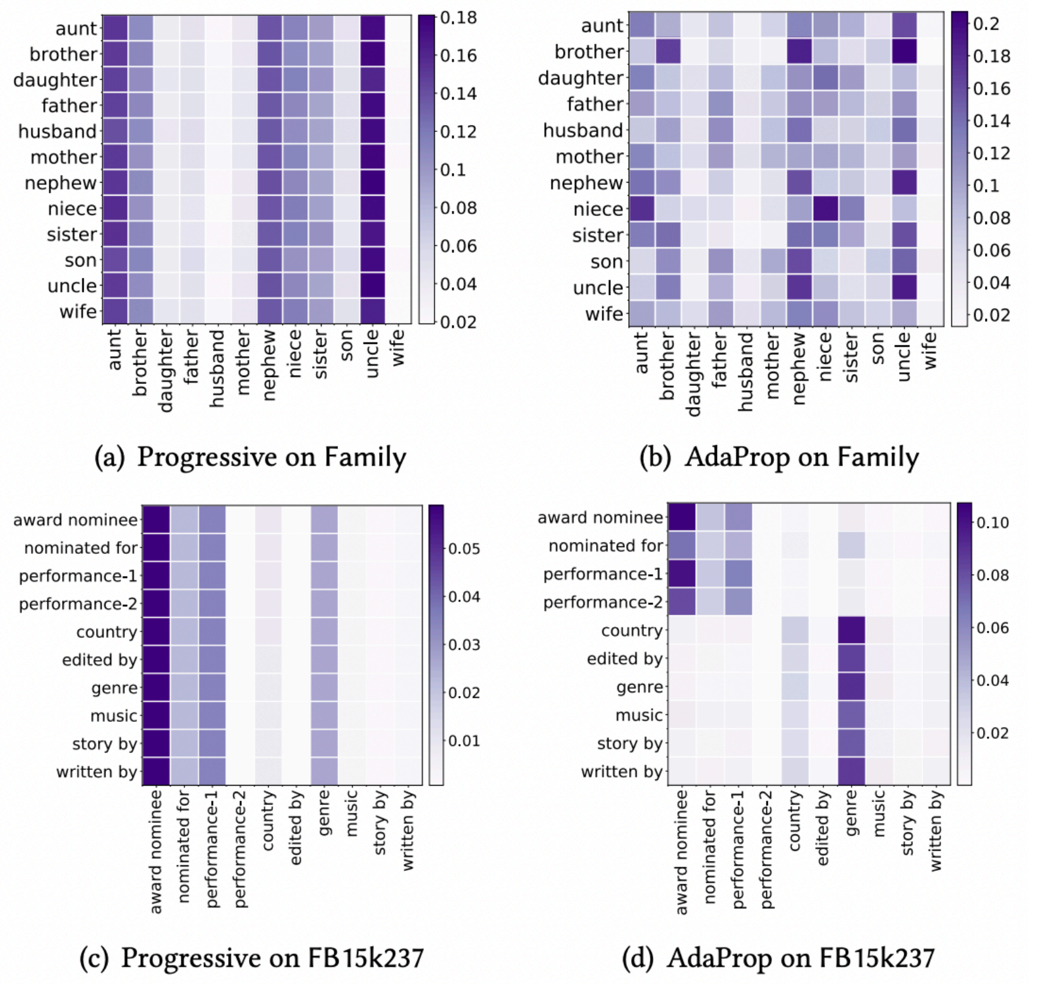
☐ Evaluation with transductive settings

type	models	Family			UMLS			WN18RR			FB15k237			NELL-995			YAGO3-10		
		MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10
non-GNN	ConvE	0.912	83.7	98.2	0.937	92.2	96.7	0.427	39.2	49.8	0.325	23.7	50.1	0.511	44.6	61.9	0.520	45.0	66.0
	QuatE	0.941	89.6	99.1	0.944	90.5	99.3	0.480	44.0	55.1	0.350	25.6	53.8	0.533	46.6	64.3	0.379	30.1	53.4
	RotatE	0.921	86.6	98.8	0.925	86.3	99.3	0.477	42.8	57.1	0.337	24.1	53.3	0.508	44.8	60.8	0.495	40.2	67.0
	MINERVA	0.885	82.5	96.1	0.825	72.8	96.8	0.448	41.3	51.3	0.293	21.7	45.6	0.513	41.3	63.7	–	–	–
	DRUM	0.934	88.1	99.6	0.813	67.4	97.6	0.486	42.5	58.6	0.343	25.5	51.6	0.532	46.0	66.2	0.531	45.3	67.6
	RNNLogic	0.881	85.7	90.7	0.842	77.2	96.5	0.483	44.6	55.8	0.344	25.2	53.0	0.416	36.3	47.8	0.554	50.9	62.2
	RLogic	–	–	–	–	–	–	0.47	44.3	53.7	0.31	20.3	50.1	–	–	–	0.36	25.2	50.4
GNNs	CompGCN	0.933	88.3	99.1	0.927	86.7	99.4	0.479	44.3	54.6	0.355	26.4	53.5	0.463	38.3	59.6	0.421	39.2	57.7
	NBFNet	0.989	**98.8**	98.9	0.948	92.0	**99.5**	0.551	49.7	66.6	0.415	32.1	**59.9**	0.525	45.1	63.9	0.550	47.9	68.6
	RED-GNN	**0.992**	**98.8**	**99.7**	0.964	94.6	99.0	0.533	48.5	62.4	0.374	28.3	55.8	0.543	47.6	65.1	0.559	48.3	68.9
	AdaProp	0.988	98.6	99.0	**0.969**	**95.6**	**99.5**	**0.562**	**49.9**	**67.1**	**0.417**	**33.1**	58.5	**0.554**	**49.3**	**65.5**	**0.573**	**51.0**	**68.5**

☐ Evaluation with inductive settings

metric	methods	WN18RR				FB15k237				NELL-995			
		V1	V2	V3	V4	V1	V2	V3	V4	V1	V2	V3	V4
Hit@10 (%)	RuleN	73.0	69.4	40.7	68.1	44.6	59.9	60.0	60.5	76.0	51.4	53.1	48.4
	Neural LP	77.2	74.9	47.6	70.6	46.8	58.6	57.1	59.3	87.1	56.4	57.6	53.9
	DRUM	77.7	74.7	47.7	70.2	47.4	59.5	57.1	59.3	87.3	54.0	57.7	53.1
	GraIL	76.0	77.6	40.9	68.7	42.9	42.4	42.4	38.9	56.5	49.6	51.8	50.6
	CoMPILE	74.7	74.3	40.6	67.0	43.9	45.7	44.9	35.8	57.5	44.6	51.5	42.1
	NBFNet	82.7	79.9	56.3	70.2	51.7	63.9	58.8	55.9	79.5	63.5	60.6	59.1
	RED-GNN	79.9	78.0	52.4	72.1	48.3	62.9	60.3	62.1	86.6	60.1	59.4	55.6
	AdaProp	**86.6**	**83.6**	**62.6**	**75.5**	**55.1**	**65.9**	**63.7**	**63.8**	**88.6**	**65.2**	**61.8**	**60.7**

☐ Heatmaps of relation type ratios in the propagation path

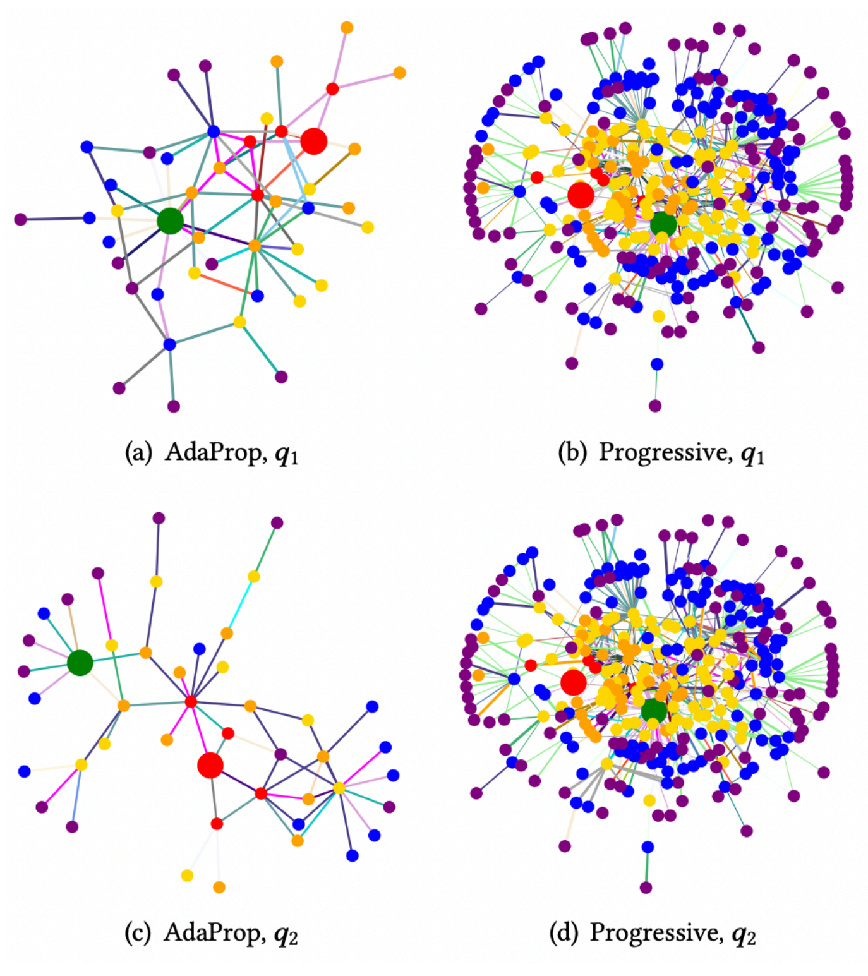


(a) Progressive on Family (b) AdaProp on Family

(c) Progressive on FB15k237 (d) AdaProp on FB15k237

semantic-aware

☐ Exemplar propagation paths on FB15k237-v1 dataset



(a) AdaProp, q_1 (b) Progressive, q_1

(c) AdaProp, q_2 (d) Progressive, q_2

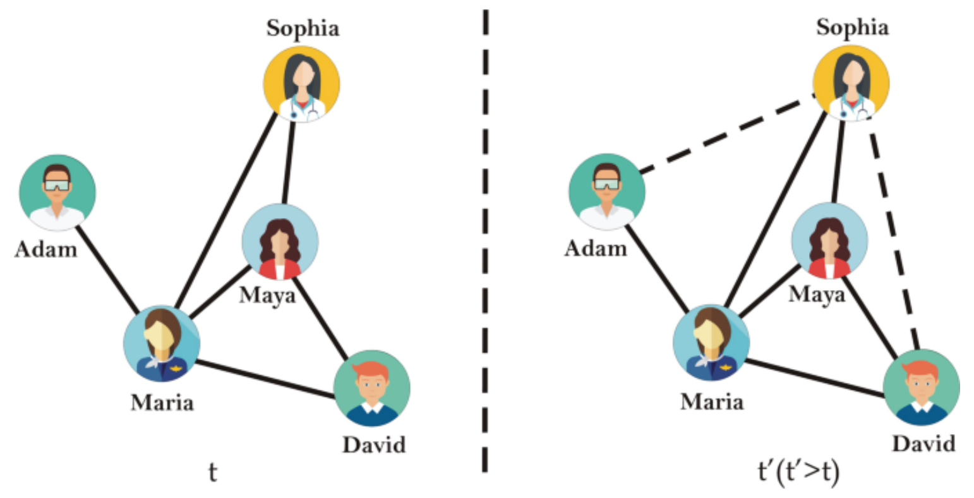
connection-preserving

Combating Bilateral Edge Noise for Robust Link Prediction

Zhanke Zhou, Jiangchao Yao, Jiaxu Liu, Xiawei Guo,
Quanming Yao, Li He, Liang Wang, Bo Zheng, Bo Han

Paper Code

Problem: Link Prediction with Noise



The link prediction task:

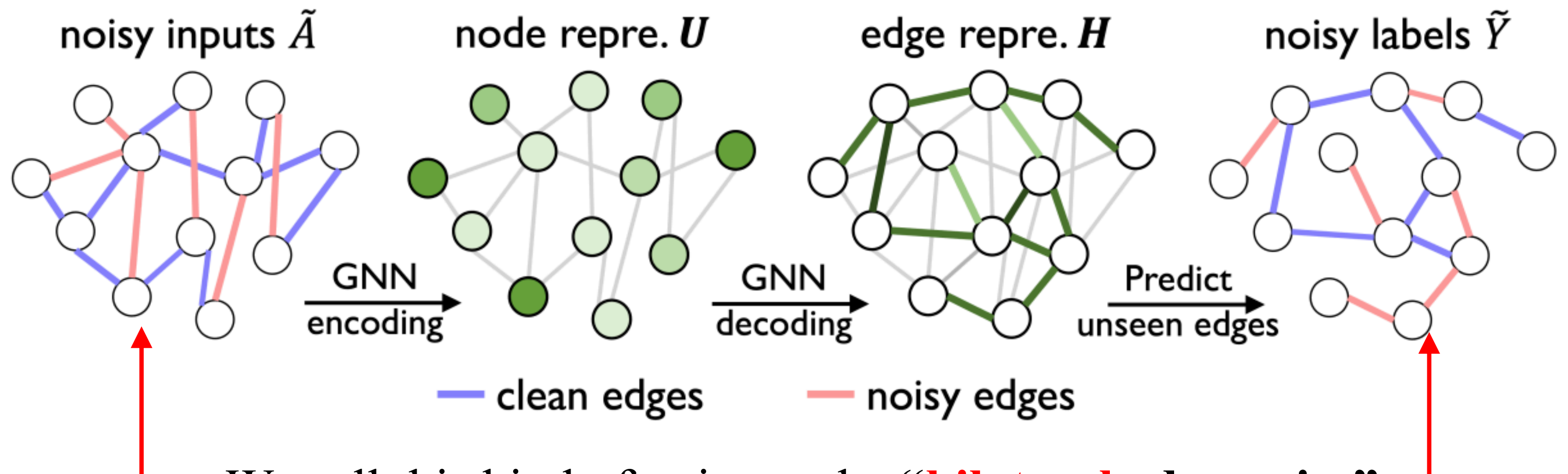
- based on the **observed** links
- to predict the **latent** links

The Bilateral Edge Noise

Existing graph benchmarks are generally **clean**.

However, graph data can be **noisy** in practical scenarios:

- the **observed** graph is often with noisy edges (**input noise**)
- the **predictive** graph often contains noisy labels (**label noise**)
- these two kinds of noise can exist at the same time (by random split)


noisy inputs $\tilde{A}$ node repre. $\boldsymbol{U}$ edge repre. $\boldsymbol{H}$ noisy labels $\tilde{Y}$

GNN encoding → GNN decoding → Predict unseen edges

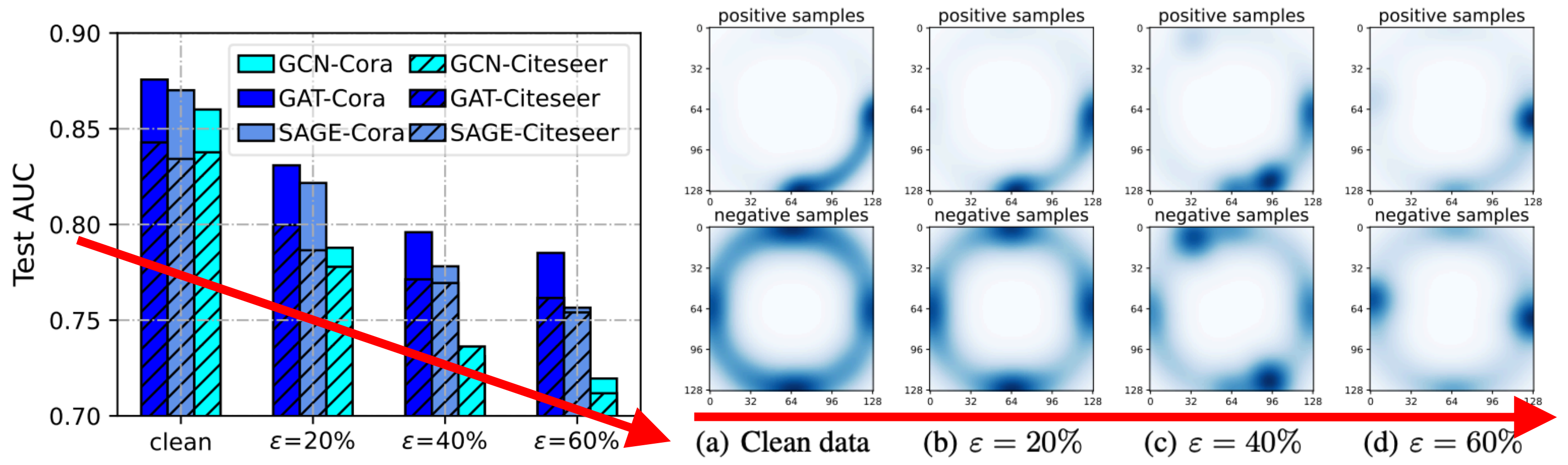
— clean edges — noisy edges

We call this kind of noise as the “**bilateral edge noise**”

Definition 3.1 (Bilateral edge noise). *Given a clean training data, i.e., observed graph $\mathcal{G}=(A,X)$ and labels $Y\in\{0,1\}$ of query edges, the noisy adjacence $\tilde{A}$ is generated by directly adding edge noise to the original adjacent matrix A while keeping the node features X unchanged. The noisy labels $\tilde{Y}$ are similarly generated by adding edge noise to the labels Y. Specifically, given a noise ratio ε_a, the noisy edges A' ($\tilde{A}=A+A'$) are generated by flipping the zero element in A as one with the probability ε_a. It satisfies that $A'\odot A=O$ and $\varepsilon_a={}^{|\texttt{nonzero}(\tilde{A})|-|\texttt{nonzero}(A)|}/_{|\texttt{nonzero}(A)|}$. Similarly, noisy labels are generated and added to the original labels, where $\varepsilon_y={}^{|\texttt{nonzero}(\tilde{Y})|-|\texttt{nonzero}(Y)|}/_{|\texttt{nonzero}(Y)|}$.*

The Effects of Bilateral Edge Noise

The noise leads to **performance degradation** and **representation collapse**:

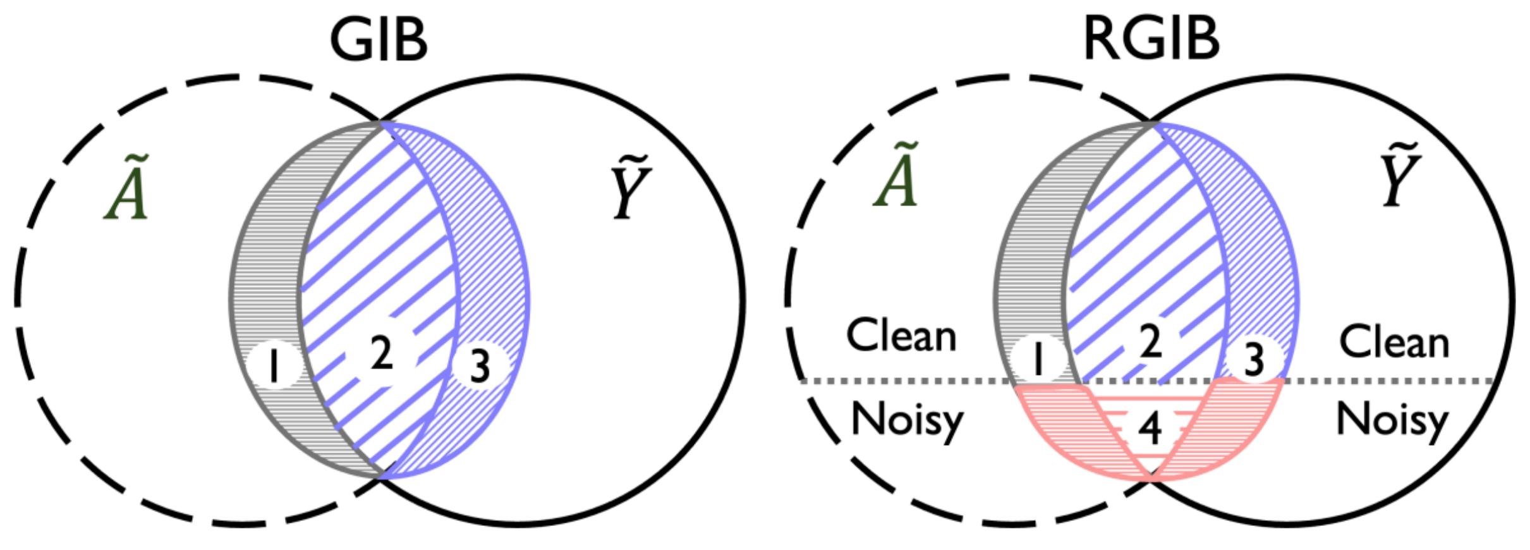


(a) Clean data (b) $\varepsilon=20\%$ (c) $\varepsilon=40\%$ (d) $\varepsilon=60\%$

performance drop **representation collapse**

➔ How to improve the robustness of GNNs under edge noise? 🤔

Method: Robust Graph Information Bottleneck



▨ optimal $\boldsymbol{H}$ ① $=I(\tilde{A};\boldsymbol{H}|\tilde{Y})$ ② $=I(\tilde{A};\tilde{Y})$ ③ $=I(\tilde{Y};\boldsymbol{H}|\tilde{A})$
④ $=I(\tilde{A};\tilde{Y}|\boldsymbol{H})$ (② + ③) $=I(\tilde{Y};\boldsymbol{H})$

$$\min \text{GIB} \triangleq -I(\boldsymbol{H};\tilde{Y}),\ \text{s.t. } I(\boldsymbol{H};\tilde{A})<\gamma,$$

➔ **GIB** is vulnerable to label noise for its maximum label supervision

In this work, we further balance the mutual dependence

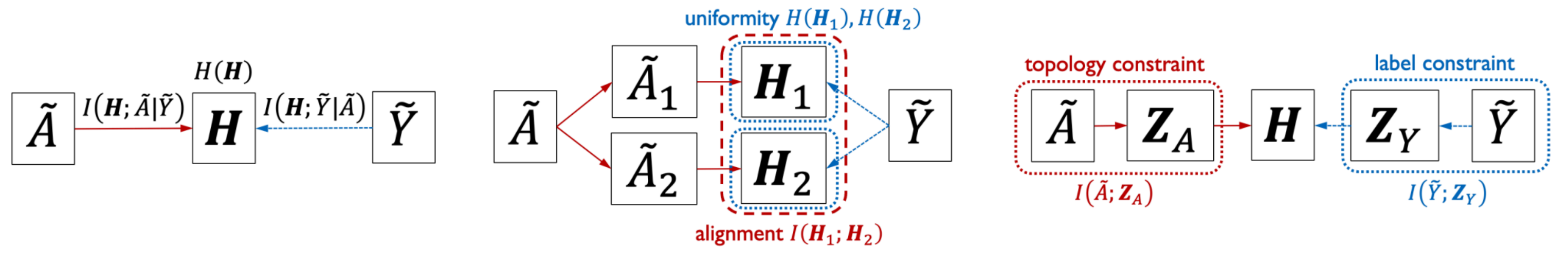
- among graph topology $\tilde{A}$, target labels $\tilde{Y}$, and representation $\boldsymbol{H}$
- build a new learning objective RGIB for robust representation

Definition 4.1 (Robust Graph Information Bottleneck). *Based on the above analysis, we propose a new learning objective to balance informative signals regarding $\boldsymbol{H}$, as illustrated in Fig. 5(a), i.e.,*

$$\min RGIB \triangleq -I(\boldsymbol{H};\tilde{Y}),\ \ s.t.\ \gamma_H^- < H(\boldsymbol{H}) < \gamma_H^+, I(\boldsymbol{H};\tilde{Y}|\tilde{A}) < \gamma_Y,\ I(\boldsymbol{H};\tilde{A}|\tilde{Y}) < \gamma_A. \qquad (2)$$

Specifically, constraints on $H(\boldsymbol{H})$ encourage a diverse $\boldsymbol{H}$ to prevent representation collapse ($>\gamma_H^-$) and also limit its capacity ($<\gamma_H^+$) to avoid over-fitting. Another two MI terms, $I(\boldsymbol{H};\tilde{Y}|\tilde{A})$ and $I(\boldsymbol{H};\tilde{A}|\tilde{Y})$, mutually regularize posteriors to mitigate the negative impact of bilateral noise on $\boldsymbol{H}$. The complete derivation of RGIB and a further comparison of RGIB and GIB are in Appendix B.2.

Instantiation: RGIB-SSL and RGIB-REP


$\tilde{A}$
$I(\boldsymbol{H};\tilde{A}|\tilde{Y})$
$H(\boldsymbol{H})$
$\boldsymbol{H}$
$I(\boldsymbol{H};\tilde{Y}|\tilde{A})$
$\tilde{Y}$
uniformity $H(\boldsymbol{H}_1), H(\boldsymbol{H}_2)$
$\tilde{A}_1$
$\tilde{A}_2$
$\boldsymbol{H}_1$
$\boldsymbol{H}_2$
alignment $I(\boldsymbol{H}_1;\boldsymbol{H}_2)$
topology constraint
$\boldsymbol{Z}_A$
$I(\tilde{A};\boldsymbol{Z}_A)$
label constraint
$\boldsymbol{Z}_Y$
$I(\tilde{Y};\boldsymbol{Z}_Y)$

RGIB **RGIB-SSL** **RGIB-REP**

$$\min \text{RGIB-SSL} \triangleq -\underbrace{\lambda_s(I(\boldsymbol{H}_1;\tilde{Y})+I(\boldsymbol{H}_2;\tilde{Y}))}_{\text{supervision}}-\underbrace{\lambda_u(H(\boldsymbol{H}_1)+H(\boldsymbol{H}_2))}_{\text{uniformity}}-\underbrace{\lambda_a I(\boldsymbol{H}_1;\boldsymbol{H}_2)}_{\text{alignment}}.$$

RGIB-SSL optimizes the representation with **self-supervised learning** to achieve a **tractable approximation** of the MI terms

- integrate a uniformity term and an alignment term with graph augmentation
- adopt the contrastive learning technique and contrast pair of samples

$$\min \text{RGIB-REP} \triangleq -\underbrace{\lambda_s I(\boldsymbol{H};\boldsymbol{Z}_Y)}_{\text{supervision}}\ \underbrace{+\lambda_A I(\boldsymbol{Z}_A;\tilde{A})+}_{\text{topology constraint}}\ \underbrace{\lambda_Y I(\boldsymbol{Z}_Y;\tilde{Y})}_{\text{label constraint}}.$$

RGIB-REP purifies the noisy signals with **reparameterization mechanism**

- latent variables $\boldsymbol{Z}_Y$ and $\boldsymbol{Z}_A$ are clean signals extracted from noisy $\tilde{Y}$ and $\tilde{A}$
- $\mathrm{I}(\boldsymbol{H};\boldsymbol{Z}_Y)$ measures the supervised signals with selected samples $\boldsymbol{Z}_Y$
- $\mathrm{I}(\boldsymbol{Z}_A;\tilde{A})$ and $\mathrm{I}(\boldsymbol{Z}_Y;\tilde{Y})$ help to select the clean information from noisy $\tilde{A}, \tilde{Y}$

Experiments

➔ RGIB performs the best in all six datasets under the bilateral noise:

method	Cora			Citeseer			Pubmed			Facebook			Chameleon			Squirrel		
	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%
Standard	.8111	.7419	.6970	.7864	.7380	.7085	.8870	.8748	.8641	.9829	.9520	.9438	.9616	.9496	.9274	.9432	.9406	.9386
DropEdge	.8017	.7423	.7303	.7635	.7393	.7094	.8711	.8482	.8354	.9811	.9682	.9473	.9568	.9548	.9407	.9439	.9377	.9365
NeuralSparse	.8190	.7318	.7293	.7765	.7397	.7148	.8908	.8733	.8630	.9825	.9638	.9456	.9599	.9497	.9402	.9494	.9309	.9297
PTDNet	.8047	.7559	.7388	.7795	.7423	.7283	.8872	.8733	.8623	.9725	.9674	.9485	.9607	.9514	.9424	.9485	.9326	.9304
Co-teaching	.8197	.7479	.7030	.7533	.7238	.7131	.8943	.8760	.8638	.9820	.9526	.9480	.9595	.9516	.9483	.9461	.9352	.9374
Peer loss	.8185	.7468	.7018	.7423	.7345	.7104	.8961	.8815	.8566	.9807	.9536	.9430	.9543	.9533	.9267	.9457	.9345	.9286
Jaccard	.8143	.7498	.7024	.7473	.7324	.7107	.8872	.8803	.8512	.9794	.9579	.9428	.9503	.9538	.9344	.9443	.9327	.9244
GIB	.8198	.7485	.7148	.7509	.7388	.7121	.8899	.8729	.8544	.9773	.9608	.9417	.9554	.9561	.9321	.9472	.9329	.9302
SupCon	.8240	.7819	.7490	.7554	.7458	.7299	.8853	.8718	.8525	.9588	.9508	.9297	.9561	.9531	.9467	.9473	.9348	.9301
GRACE	.7872	.6940	.6929	.7632	.7242	.6844	.8922	.8749	.8588	.8899	.8865	.8315	.8978	.8987	.8949	.9394	.9380	.9363
RGIB-REP	.8313	.7966	.7591	.7875	.7519	.7312	.9017	.8834	.8652	**.9832**	**.9770**	.9519	**.9723**	**.9621**	**.9519**	**.9509**	**.9455**	**.9434**
RGIB-SSL	**.8930**	**.8554**	**.8339**	**.8694**	**.8427**	**.8137**	**.9225**	**.8918**	**.8697**	.9829	.9711	**.9643**	.9655	.9592	.9500	.9499	.9426	.9425

➔ RGIB consistently surpasses all the baselines under the unilateral noise:

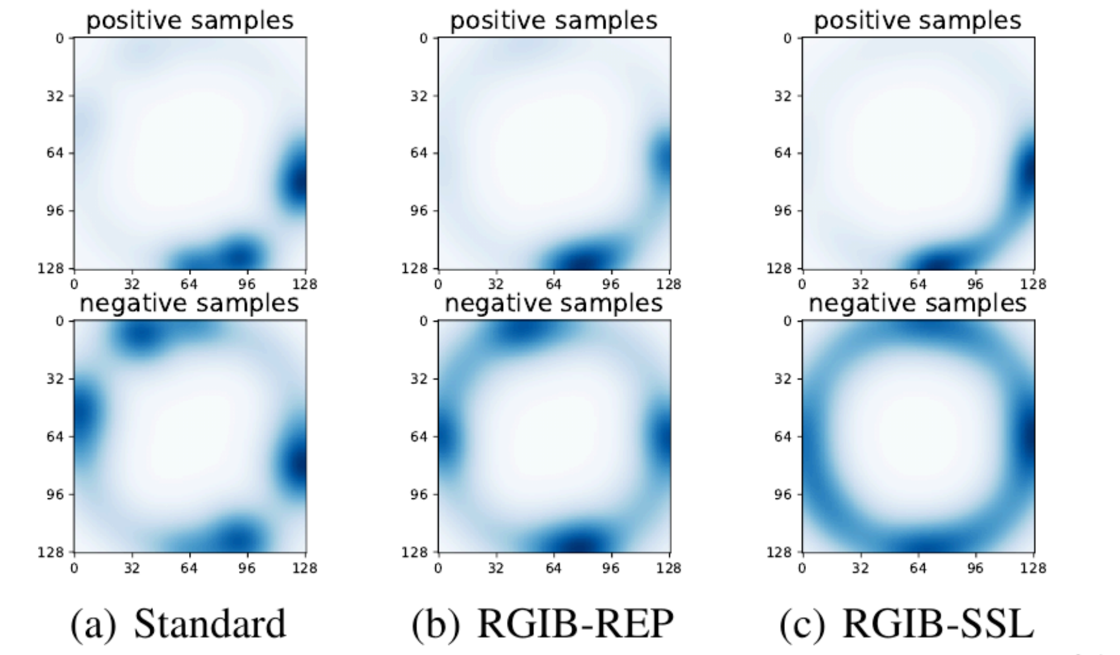
input noise	Cora			Citeseer			Pubmed			Facebook			Chameleon			Squirrel		
	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%
Standard	.8027	.7856	.7490	.8054	.7708	.7583	.8854	.8759	.8651	.9819	.9668	.9622	.9608	.9433	.9368	.9416	.9395	.9411
DropEdge	.8338	.7826	.7454	.8025	.7730	.7473	.8682	.8456	.8376	.9803	.9685	.9531	.9567	.9433	.9432	.9426	.9376	.9358
NeuralSparse	.8534	.7794	.7637	.8093	.7809	.7468	.8931	.8720	.8649	.9712	.9691	.9583	.9609	.9540	.9348	.9469	.9403	.9417
PTDNet	.8433	.8214	.7770	.8119	.7811	.7638	.8903	.8776	.8609	.9725	.9668	.9493	.9610	.9457	.9360	.9469	.9400	.9379
Co-teaching	.8045	.7871	.7530	.8059	.7753	.7668	.8931	.8792	.8606	.9712	.9707	.9714	.9524	.9446	.9447	.9462	.9425	.9306
Peer loss	.8051	.7866	.7517	.8106	.7767	.7653	.8917	.8811	.8643	.9758	.9703	.9622	.9558	.9482	.9412	.9362	.9386	.9336
Jaccard	.8200	.7838	.7617	.8176	.7776	.7725	.8987	.8764	.8639	.9784	.9702	.9638	.9507	.9436	.9364	.9388	.9345	.9240
GIB	.8002	.8099	.7741	.8070	.7717	.7798	.8932	.8808	.8618	.9796	.9647	.9650	.9605	.9521	.9416	.9390	.9406	.9397
SupCon	.8349	.8301	.8025	.8076	.7767	.7655	.8867	.8739	.8558	.9647	.9517	.9401	.9606	.9536	.9468	.9372	.9343	.9305
GRACE	.7877	.7107	.6975	.7615	.7151	.6830	.8810	.8795	.8593	.9015	.8833	.8395	.8994	.9007	.8964	.9392	.9378	.9363
RGIB-REP	.8624	.8313	.8158	.8299	.7996	.7771	.9008	.8822	.8687	**.9833**	**.9723**	**.9682**	**.9705**	**.9604**	.9480	**.9495**	**.9432**	.9405
RGIB-SSL	**.9024**	**.8577**	**.8421**	**.8747**	**.8461**	**.8245**	**.9126**	**.8889**	**.8693**	.9821	.9707	.9668	.9658	.9570	**.9486**	.9479	.9429	**.9429**

label noise	Cora			Citeseer			Pubmed			Facebook			Chameleon			Squirrel		
	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	60%
Standard	.8281	.8054	.8060	.7965	.7850	.7659	.9030	.9039	.9070	.9882	.9880	.9886	.9686	.9580	.9362	.9720	.9720	.9710
DropEdge	.8363	.8273	.8148	.7937	.7853	.7632	.9313	.9201	.9240	.9673	.9771	.9776	.9580	.9579	.9578	.9608	.9603	.9698
NeuralSparse	.8524	.8246	.8211	.7968	.7921	.7752	.9272	.9136	.9089	.9781	.9781	.9784	.9583	.9583	.9571	.9633	.9626	.9625
PTDNet	.8460	.8214	.8138	.7968	.7765	.7622	.9219	.9099	.9093	.9879	.9880	.9783	.9585	.9576	.9665	.9633	.9623	.9626
Co-teaching	.8446	.8209	.8157	.7974	.7877	.7913	.9315	.9291	.9319	.9762	.9797	.9638	.9642	.9650	.9533	.9675	.9641	.9655
Peer loss	.8325	.8036	.8069	.7991	.7990	.7751	.9126	.9101	.9210	.9769	.9750	.9734	.9621	.9501	.9569	.9636	.9694	.9696
Jaccard	.8289	.8064	.8148	.8061	.7887	.7689	.9098	.9135	.9096	.9702	.9725	.9758	.9603	.9659	.9557	.9529	.9512	.9501
GIB	.8337	.8137	.8157	.7986	.7852	.7649	.9037	.9114	.9064	.9742	.9703	.9771	.9651	.9582	.9489	.9641	.9628	.9601
SupCon	.8491	.8275	.8256	.8024	.7983	.7807	.9131	.9108	.9162	.9647	.9567	.9553	.9584	.9580	.9477	.9516	.9595	.9511
GRACE	.8531	.8237	.8193	.7909	.7630	.7737	.9234	.9252	.9255	.8913	.8972	.8887	.9053	.9074	.9075	.9171	.9174	.9166
RGIB-REP	.8554	.8318	.8297	.8083	.7846	.7945	.9357	.9343	.9332	**.9884**	**.9883**	**.9889**	**.9785**	**.9797**	**.9785**	**.9735**	**.9733**	**.9737**
RGIB-SSL	**.9314**	**.9224**	**.9241**	**.9204**	**.9218**	**.9250**	**.9594**	**.9604**	**.9613**	.9857	.9881	.9857	.9730	.9752	.9744	.9727	.9729	.9726

➔ the graph representation has obvious improvement in distribution:

Table 5: Comparison of alignment. Here, std. is short for *standard training*, and SSL/REP are short for RGIB-SSL/RGIB-REP, respectively.

dataset	Cora			Citeseer		
method	std.	REP	SSL	std.	REP	SSL
clean	.616	.524	**.475**	.445	.439	**.418**
$\varepsilon=20\%$	.687	.642	**.543**	.586	.533	**.505**
$\varepsilon=40\%$	.695	.679	**.578**	.689	.623	**.533**
$\varepsilon=60\%$	.732	.704	**.615**	.696	.647	**.542**



(a) Standard (b) RGIB-REP (c) RGIB-SSL

Figure 6: Uniformity distribution on Citeseer with $\varepsilon=40\%$.

Future Directions

New instantiations of RGIB, e.g., approximation of the MI terms
New scenarios with noise, e.g., feature noise, other structural noise
New graph learning tasks, e.g., node classification, graph classification
New theoretical analysis, e.g., information theory, graph generation model