SUREL+: Moving from Walks to Sets for Scalable Subgraph-based Graph Representation Learning

Haoteng Yin[†], Muhan Zhang[‡], Jianguo Wang[†], Pan Li[†§]
[†]Department of Computer Science, Purdue University [‡] Institute for Artificial Intelligence, Peking University
[§]School of Electrical and Computer Engineering, Georgia Institute of Technology
[†]{yinht, csjgwang, panli}@purdue.edu [‡]muhan@pku.edu.cn

ABSTRACT

Subgraph-based graph representation learning (SGRL) has recently emerged as a powerful tool in many prediction tasks on graphs due to its advantages in model expressiveness and generalization ability. Most previous SGRL models face computational issues associated with the high cost of extracting subgraphs for each training or testing query. Recently, SUREL has been proposed as a new framework to accelerate SGRL, which samples random walks offline and joins these walks as subgraphs online for prediction. Due to the reusability of sampled walks across different queries, SUREL achieves state-of-the-art performance in both scalability and prediction accuracy. However, SUREL still suffers from high computational overhead caused by node redundancy in sampled walks. In this work, we propose a novel framework SUREL+ that upgrades SUREL by using node sets instead of walks to represent subgraphs. This set-based representation avoids node duplication by definition, but the sizes of node sets can be irregular. To address this issue, we design a dedicated sparse data structure to efficiently store and fast index node sets, and provide a specialized operator to join them in parallel batches. SUREL+ is modularized to support multiple types of set samplers, structural features, and neural encoders to complement the loss of structural information due to the reduction from walks to sets. Extensive experiments have been performed to validate SUREL+ in the prediction tasks of links, relation types, and higher-order patterns. SUREL+ achieves 3-11× speedups of SUREL while maintaining comparable or even better prediction performance; compared to other SGRL baselines, SUREL+ achieves ~20× speedups and significantly improves the prediction accuracy.

PVLDB Reference Format:
Haoteng Yin[†], Muhan Zhang[‡], Jianguo Wang[†], Pan Li[†§]. SUREL+: Moving from Walks to Sets for Scalable Subgraph-based Graph Representation Learning. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Graph-COM/SUREL_Plus.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Graphs are widely used to model interactions in natural sciences and relationships in social life [20, 24]. Graph-structured data in the real world are highly irregular and often large-scale. To solve inference tasks on graphs, graph representation learning (GRL) that studies quantitative representations of graph-structured data has attracted much attention [15, 16, 49]. Recently, subgraph-based GRL (SGRL) has become an important research direction for researchers studying GRL algorithms and systems, as it has achieved far better prediction performance than other approaches in many GRL tasks, especially those involving a set of nodes. Given a set of nodes of interest, namely a queried node set, SGRL models such as SEAL [54, 57], GraIL [41], and SubGNN [1] first extract a subgraph around the queried node set (termed query-induced subgraph), and then encode the extracted subgraph for prediction. Extensive works have shown that SGRL models are more robust [52] and more expressive [5, 12]; while canonical graph neural networks (GNNs) including GCN [23] and GraphSAGE [14] generally fail to make accurate predictions, due to their limited expressive power [9, 13, 57], incapability of capturing intra-node distance information [28, 39], and improper entanglement between receptive field size and model depth [18, 51, 52]. An example in Fig. 1 illustrates how SGRL works for link prediction and demonstrates its advantages over canonical GNNs that generate and aggregate node representations to predict links. Here, canonical GNNs would map nodes in structural symmetry into the same representation and cause the ambiguity issue [50, 57]. So far, the advantages of SGRL methods have been proved in many applications, such as link and relation prediction [41, 54, 57], higher-order pattern prediction [30, 33], temporal network modeling [48], recommender systems [55], anomaly detection [1, 6], graph meta-learning [18], subgraph matching [29, 31], and molecular/protein research in life sciences [38, 47].

Albeit with multiple benefits of its algorithm, current SGRL models face two major computational challenges that hinder their deployment in practice: (1) **Query Dependency**. A subgraph must be extracted for each queried node set, which is not reusable across different queries and cannot be preprocessed if the query is unknown; (2) **Irregularity**. The extracted subgraphs are irregularly sized, resulting in performance degradation in batch processing and load balancing. As shown in Fig. 3 (a), subgraph extraction in SEAL [54, 56] is extremely slow and expensive. This has inspired recent work on dedicated hardware acceleration for subgraph extraction [11, 36]. However, how to improve the scalability and efficiency of SGRL methods remains largely undeveloped.

SUREL [51] is the state-of-the-art (SOTA) framework that applies algorithm and system co-design to implement SGRL. It employs

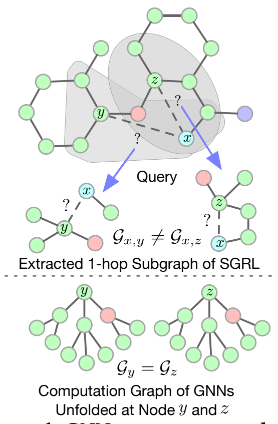

Query
$\mathcal{G}_{x,y} \neq \mathcal{G}_{x,z}$
Extracted 1-hop Subgraph of SGRL
$\mathcal{G}_y = \mathcal{G}_z$
Computation Graph of GNNs Unfolded at Node y and z

Figure 1: GNNs cannot correctly predict whether x is more likely linked with y or z, because y and z have the same node representations. However, the representations based on one-hop neighbors are more expressive to distinguish the two node pairs.

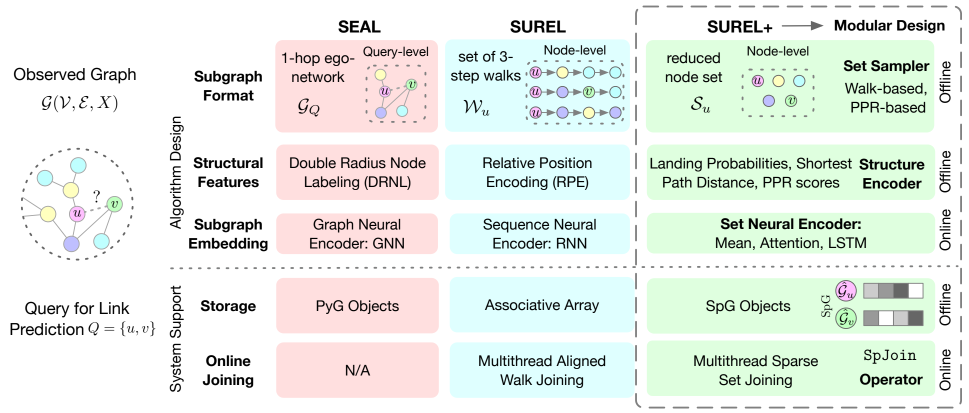

Observed Graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$
Query for Link Prediction $Q = \{u, v\}$
SEAL
SUREL
SUREL+
Modular Design
Algorithm Design
System Support
Subgraph Format
1-hop ego-network $\mathcal{G}_Q$
Query-level
set of 3-step walks $\mathcal{W}_u$
Node-level
reduced node set $\mathcal{S}_u$
Node-level
Set Sampler
Walk-based, PPR-based
Offline
Structural Features
Double Radius Node Labeling (DRNL)
Relative Position Encoding (RPE)
Landing Probabilities, Shortest Path Distance, PPR scores
Structure Encoder
Offline
Subgraph Embedding
Graph Neural Encoder: GNN
Sequence Neural Encoder: RNN
Set Neural Encoder: Mean, Attention, LSTM
Online
Storage
PyG Objects
Associative Array
SpG Objects
SpG
Offline
Online Joining
N/A
Multithread Aligned Walk Joining
Multithread Sparse Set Joining
SpJoin Operator
Online

Figure 2: Overview of SUREL+: Side-by-Side Comparison with Previous SGRL Methods via an Example for Link Prediction. SUREL+ samples node sets while SEAL [54, 57] extracts the whole subgraph for each query and SUREL [51] samples walks. To serve node set-based representations, SUREL+ designs a new algorithm with dedicated system support. SUREL+ provides various types of set samplers, structure encoders, and set neural encoders to complement the loss of structural information by reducing subgraphs to node sets. SUREL+ also builds on a customized sparse data structure `SpG` to store sampled node sets for fast access, and supports online set joins in parallel via sparse arithmetic operator `SpJoin`.

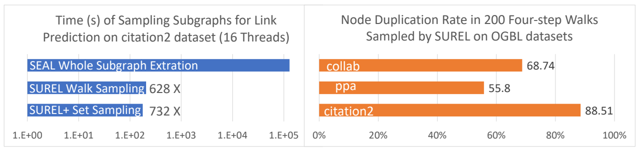


Figure 3: (a) Subgraph extraction in SEAL [54, 57] has much higher complexity compared to other samplers with simplified forms and suffers from the irregularity of subgraphs and "neighborhood explosion" [8, 52]. (b) Breaking subgraphs into reusable walks reduces complexity but also faces the issue of heavy node duplicates.

reusable node-level sampled walks to represent query-specific subgraphs. Specifically, SUREL regards each node in the graph as a seed and runs a group of random walks from the seed offline. Given a queried node set, SUREL online joins and encodes the sampled walks for all nodes in the query for prediction. The joint walks here essentially act as a proxy of the subgraph for the queried node set. To compensate for the loss of structural information in subgraphs represented by walks, one type of structural feature, termed relative position encoding (RPE), is adopted to locate each node in the subgraph. RPEs are pre-computed offline and then attached to walks before feeding into neural networks (NNs) to make predictions. In SUREL, sampled walks from one seed can be reused across multiple queries whenever that seed node is involved. SUREL significantly improves the scalability of SGRL by this walk-sharing mechanism and benefits from the regularity of walks, which enables highly parallel walk sampling and online joining through dedicated system design. However, SUREL still faces many inherent drawbacks of adopted walk-based representations, i.e., high node duplication rate in sampled walks (over 55%, see Fig. 3 (b)). This further raises the following computational issues: (1) **extra space cost** of hosting walks in memory, (2) **high workload** of data transfer from CPU to GPU, and (3) **extra time consumption** of redundant operations in joining walks and NN-based encoding in subsequent routines.

In this work, we upgrade SUREL and develop a novel framework SUREL+ that benefits from algorithm-system co-design once more. The key idea of SUREL+ is simple: it evolves from using walks to using node sets to represent subgraphs, thereby obviating node duplication. However, this new node set-based idea brings many difficulties in algorithm and system design. Regarding the algorithm side, reducing from whole subgraphs to walks and further to node sets studied in this work loses a considerable amount of structural information. Consequently, developing an algorithm that can compensate for such loss while maintaining performance is a key challenge for algorithm design. On the system side, the use of walks in SUREL [51] allows for easy storage and processing in a regular, aligned format by controlling the length and the number of walk sampling. In contrast, node sets present irregular sizes, making the efficient storage and access of these sets a daunting task. Particularly, how to coordinate the designs of both sides constitutes the main challenge of this work.

SUREL+ addresses the above challenges throughout its whole pipeline. During preprocessing, SUREL+ offline extracts a subset of unique nodes from the neighborhood of each node in the graph by sampling. To compensate for the loss of structural information, SUREL+ incorporates various types of *set samplers* and *structure encoders* to construct structural features for each sampled node. Specifically, set samplers leverage different graph metrics to measure node importance and determine sampling rules. Structure encoders utilize landing probabilities of random walks [27], shortest path distances, and personalized PageRank scores [19], which broadly covers the structural features adopted by previous SGRL methods [28, 41, 47, 51, 54, 57]. Furthermore, SUREL+ designs a dedicated sparse data structure, namely `SpG`, which stores sampled node sets in a memory-efficient manner and allow fast access. During training and inference, SUREL+ online joins all the sampled node sets and their associated structural features belonging to seed nodes in a given query, to represent the query-induced subgraph and make predictions. SUREL+ also provides a specialized operator `SpJoin` that improves join operations on node sets in parallel batches with provably fast speed. In addition, to capture different levels of interactions between structural features, SUREL+ supports multiple *set neural encoders*, such as multi-linear perception + mean pooling, set attention [42] and LSTM [14] that ensure consistently good performance across different types of SGRL tasks.

Overall, our contributions can be summarized as follows:

- Algorithm: SUREL+ is a novel SGRL framework (*open source*), that utilizes reusable node sets and associates them with various types of structural features to represent query-induced subgraphs through their online joining. Adopting sets in SUREL+ greatly saves memory and computation but without degrading prediction performance compared with the SOTA baselines.
- System: SUREL+ designs a dedicated sparse data structure `SpG` and an arithmetic operator `SpJoin` to support efficient storage and processing of node sets, which achieves much better efficiency and scalability than previous SGRL methods.
- We conduct extensive experiments on 7 real-world graphs, with millions of nodes/edges, and demonstrate the advantages of SUREL+ in link/relation-type/motif prediction tasks. SUREL+ is 3-11× faster than the current SOTA SGRL method SUREL while maintaining comparable or even better prediction accuracy. SUREL+ also achieves ~20× speedup with substantial prediction accuracy improvements over other SGRL baselines.

2 PRELIMINARIES

2.1 Notations and Relevant Definitions in SGRL

Let $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ be an attributed graph with node set $\mathcal{V} = \{1, 2, ..., n\}$ and edge set $\mathcal{E}$, where $X \in \mathbb{R}^{n \times d}$ denotes node attributes with d-dimension. A query $Q \subset \mathcal{V}$ is a node set of interest for a certain type of task. We denote the subgraph induced by query Q as $\mathcal{G}_Q$ and the node-induced subgraph as $\mathcal{G}_u$, where induced subgraphs are typically within a small number of hops.

Definition 2.1 (Subgraph-based Graph Representation Learning (**SGRL**))**.** Given a query Q of node set over graph $\mathcal{G}$, SGRL aims to learn a representation of the query-induced subgraph $\mathcal{G}_Q$ to make prediction $f(\mathcal{G}_Q)$. $(\cdot)$ is usually a neural network. SGRL tasks come with some labeled queries $\{(Q_i, y_i)\}_{i=1}^{L}$ for supervision and other unlabeled queries $\{Q_i\}_{i=L+1}^{L+N}$ for inference.

Examples of SGRL Tasks *Link prediction* seeks to estimate the likelihood of a link between two endpoints in a given graph, where a query Q corresponds to a node pair. It can be further generalized to predict links with types over heterogeneous graphs [41] or to predict vascular access [34] and chemical bond [20] for domain-specific graphs. Tasks beyond pairwise relations are named *higher-order pattern prediction*, where a query Q consists of three or more nodes. In this work, we consider that given partially observed pairwise relations among queried nodes in Q, whether these nodes will establish certain full higher-order relation of interest [30, 40].

Review of Current SGRL Pipelines The SGRL framework has three main parts, as shown in the *Algorithm Design* section in Fig. 2: preparation of subgraphs, construction of structural features, and neural encoding to obtain embeddings. Both subgraph extraction and structural feature depend on queries, so these two steps are often coupled with queries in classical SGRL models, e.g., SEAL [54, 57]. However, such coupling is expensive and makes the underlying computation impossible to share between queries, which motivates recent SGRL methods to decouple them. SUREL [51] replaces explicit subgraph extraction with the online joining of multiple node-level walks that can be pre-sampled offline. In addition, relative position encoding defined on sampled walks is used as structural features. These sampled walks and associated structural features can be shared and reused to assemble subgraphs for multiple queries, which improves the reusability and model scalability. Neural networks are then applied to encode and aggregate these walks for prediction.

2.2 Related Works

Scalable GNN Design. GNNs are currently the most widely used tool for GRL, though they suffer from some issues when applied directly as SGRL models. To enhance the scalability of GNNs, current studies mainly focus on improving graph subsampling and mini-batch training techniques[10, 53]. But the graph subsampling used in GNNs differs fundamentally from the subgraph extraction step in SGRL: the former aims to tackle the GPU memory overflow issue during full-batch training; the latter is a key part of SGRL algorithm used to sample subgraphs around queried node sets of interest as features to make predictions over. With different goals, scaling techniques for GNNs are not applicable to SGRL models. Several distributed GNN systems have been purposed to deal with industrial-level graphs by reducing communication and synchronization costs through specialized system design such as pipelining [45], partitioned parallelism [44] and update with staleness [35]. Unfortunately, these methods are not directly applicable to address the challenges faced by SGRL methods, such as subgraph extraction.
Scalable SGRL Design. Efficient subgraph extraction is the main direction of recent system works to scale SGRL models. These techniques include PPR-based [4, 52] and random walk-based [51] subgraph samplers, node neighborhood sampling through CUDA kernel (DGL, [11]), tensor operations (PyG, [36]), and performance-engineered sampler (SALIENT, [21]), as well as parallel sampling for temporal graphs [58]. Some frameworks also customize data structures to better support subgraph operations and gain higher throughput, such as associative arrays in SUREL [51], temporal-CSR in TGL [58] and GPU-orientated dictionary in NAT [32]. To achieve scalable modeling design, ShaDow [52] restricts receptive fields to localized subgraphs and thus decouples the depth and scope of GNNs. GDGNN [25] employs node representations along geodesic paths for predictions to avoid complex structural feature construction. BUDDY [7] uses subgraph sketches for link prediction without explicitly constructing query-induced subgraphs. However, these works partially address SGRL's scalability issues (bottleneck of extraction, storage, or feature construction) or are limited to specific tasks e.g. link prediction. In contrast, SUREL+ provides a general subgraph-based framework through comprehensive co-design in scalable sampling, efficient storage, and expressive modeling.

3 THE FRAMEWORK OF SUREL+

This section introduces SUREL+, whose key concept is to pre-sample node sets offline from the neighborhoods of nodes in the graph, which can be joined online as a proxy for query-induced subgraphs. This approach allows the reuse of sampled node sets across different queries, and its set-based representation also resolves memory and computation issues caused by node duplication in existing walk-based representations as SUREL [51] adopted. SUREL+ has a modular design, supports different set samplers, structure encoders to construct various structural features, and multiple set neural encoders, which can be flexibly selected to compensate for the loss of structural information after sampling. SUREL+ also designs a dedicated sparse data structure SpG and an arithmetic operator SpJoin for efficient storage and online batch-wise set joins. Fig. 2 provides a comparison between SUREL+ and current SGRL methods. The following subsections provide further details on these modules.

3.1 Set Samplers and Structure Encoders

SUREL+ uses set samplers to sample a set of nodes from the neighborhood of each node in the graph and calls structure encoders to construct structural features. Both of these operations are executed offline. The former is primarily for computational benefits, while the latter is performed to offset the loss of structural information due to the reduction from subgraphs (adopted by SEAL [54, 57]) or walks (adopted by SUREL [51]) to sets. Conceptually, SUREL+ represents the node-induced subgraph $\mathcal{G}_u$ via a combination of (a) a unique node set $\mathcal{S}_u$ comprising sampled nodes from the neighborhood of node u and (b) the associated structural features $\mathcal{Z}_u$ that reflects the position in $\mathcal{G}_u$ of each sampled node in $\mathcal{S}_u$.

Set Samplers Two types of set samplers are adopted. The first type named *Walk-based Sampler* is to sample short-step random walks and eliminate duplicate nodes during sampling. The second type named *Metric-based Sampler* is based on more principled graph metrics that measure the proximity between nodes and the seed node, such as personalized PageRank (PPR) scores [19] or short path distances. Specifically, the walk-based sampler runs M-many m-step random walks, starting from each seed node u in parallel on the graph $\mathcal{G}$, and then puts only unique nodes on these walks into the set $\mathcal{S}_u$. The metric-based sampler, taking PPR-based [4] as an example, first runs the push-flow algorithm [2] to obtain an approximation of the PPR vector for each seed node u, and then selects the top-K nodes with the highest PPR scores into the set $\mathcal{S}_u$. Mathematically, PPR scores are convergent landing probabilities of seeded random walks that reach infinite steps. Therefore, the two samplers complement each other by leveraging either more local or global structures of the graph. We use hyper-parameters M, m to control random walks, and K to control metric-based sampler, which are all set as some constants in practice. The complexity of the above offline sampling procedures is $O(|\mathcal{V}|)$.

Structure Encoders The structure encoder is to construct structural features $\mathcal{Z}_{u,x} \in \mathbb{R}^k$ for each node x in the sampled node set $\mathcal{S}_u$. These features are provably crucial for inference tasks involving multiple nodes [57], and can be conceptually understood as defining the position of node x in relation to the anchor node u within its neighborhood. One possible choice is landing probabilities (LPs) of random walk [27, 28, 51]: each element $\mathcal{Z}_{u,x}[i]$ stores the counts of node x landed at step i of all walks sampled by the walk-based sampler starting at u divided by the total number of walks. By definition, landing probabilities can be obtained along with the walk sampling procedure. Another option is the shortest path distance (SPD) between x and u [28, 54, 57]. PPR scores [19] are also useful structural features and can be computed during the metric-based sampler runs. Later, we denote the group of structural features for all nodes in $\mathcal{S}_u$ as $\mathcal{Z}_u = \{\mathcal{Z}_{u,x} | x \in \mathcal{S}_u\}$.

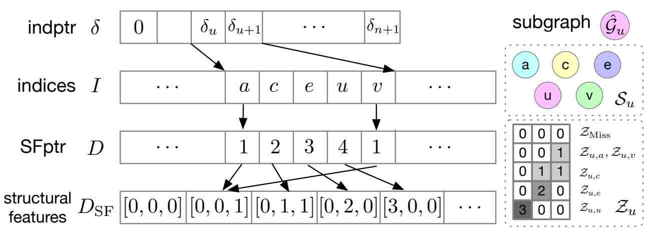


Figure 4: Node set $\mathcal{S}_u$ and its associated structural features $\mathcal{Z}_u$ stored in SpG. Here, D_{SF} keeps the counts of nodes landed at different steps in sampled walks, which can be normalized later to landing probabilities as structural features, and is shown as an example.

3.2 Set-based Storage - SpG

Node-set-based representations have advantages in terms of reusability and elimination of duplicates. However, due to the uneven size of sampled node sets, their storage and fast access pose great challenges. Note that these node sets need to be visited frequently during subsequent online phases for prediction purposes. To overcome these obstacles, SUREL+ devises a specialized compressed sparse row (CSR) format called SpG, which reorganizes the storage of node sets and their structural features in a memory-efficient manner. As illustrated in Fig. 4, the node set $\mathcal{S}_u$ and its structural features $\mathcal{Z}_u$ are stored in a row of SpG, denoted as SpG$[u, :]$. Multiple node sets and their respective structural features are consolidated into three contiguous arrays:

- `indptr` $\delta \in \mathbb{Z}^{n+1}$, an integer array tracks the starting index of each stored node set (row). It records the cumulative sum of the sizes of all node sets $\mathcal{S}_u$, $\forall u \in \mathcal{V}$, i.e., $\delta_{u+1} = \delta_u + |\mathcal{S}_u|$, where $|\mathcal{S}_u|$ represents the size of the node set $\mathcal{S}_u$. The total number of nodes stored in SpG is δ_{n+1};
- `indices` $I \in \mathbb{Z}^{\delta_{n+1}}$, a coalesce array of all node sets $\mathcal{S}_u$, $\forall u \in \mathcal{V}$. The segment $I[\delta_u : \delta_{u+1}]$ corresponds to indices of sampled nodes of $\mathcal{S}_u$ stored in sorted order. This ordering is useful for accelerating the join operations discussed in Sec. 3.3.
- `SFptr` $D \in \mathbb{R}^{\delta_{n+1}}$, an pointer array contains the indices of the structural features $\mathcal{Z}_u$ stored in the array D_{SF}. D_{SF} is designed to remove duplicated structural features by hashing and usually resides in GPU memory. This secondary index can further improve memory efficiency, especially when using LPs/SPDs which likely contain a large number of duplicates. It is not necessary when using PPR scores as their values do not have many duplicates.

Regarding the cost of SpG, `indptr` array is of size $|\mathcal{V}| + 1$, and the size of both `indices` and `SFptr` arrays is δ_{n+1}. The compressed encoding array D_{SF} has a size of $c * k$, where c is the number of unique structural features stored and k denotes feature dimension. The total complexity of this data structure is $O(|\mathcal{V}| + \delta_{n+1} + c * k)$.

Comparison with Other Methods Table 1 summarizes the space complexity comparison. By adopting the walk-based sampler (sampling M-many m-step walks), δ_{n+1} is about one-fifth of $mM|\mathcal{V}|$ used by SUREL, while the metric-based sampler (sampling top-K PPR scores) results in $\delta_{n+1} = K|\mathcal{V}|$. Both values are substantially lower than $O(S|\mathcal{E}|)$ used by SEAL, where S is the average size of sampled subgraphs. For hosting structural features, SUREL+ adopts the secondary index `SFptr` D and only retains the unique structural features in D_{SF} to further reduce memory footprint. Since

Table 1: Space Complexity Comparison. Suppose using $O(|\mathcal{E}|)$-many queries and S to denote the average size of sampled subgraphs for SGRLs. d and k are respective dimensions of node and structural features. M-many m-step walks are used for sampling in SUREL and SUREL+ (with the walk-based sampler). c is the number of unique k-dim structural features. δ_{n+1} is the sum of the sizes of node sets.

Methods	GNN [23]	SEAL [54, 57]	SUREL [51]	SUREL+
Structure	$O(\lvert\mathcal{V}\rvert + \lvert\mathcal{E}\rvert)$	$O(S\lvert\mathcal{E}\rvert)$	$O(mM\lvert\mathcal{V}\rvert)$	$O(\delta_{n+1})$
Feature	$O(d\lvert\mathcal{V}\rvert)$	$O(kS\lvert\mathcal{E}\rvert)$	$O(kmM\lvert\mathcal{V}\rvert)$	$O(\delta_{n+1} + c * k)$

c is typically independent of $|\mathcal{V}|$ in practice, SUREL+ with SpG is well-suited for handling large graphs.

3.3 Joining Node Sets via Sparse Operations

The goal of joining node sets is to construct a proxy for query-induced subgraphs from pre-sampled node sets to make predictions for SGRL queries. For a given query Q, we *join* relevant node sets $\mathcal{S}_u, \forall u \in Q$ into $\mathcal{S}_Q = \bigcup_{u \in Q} \mathcal{S}_u$ and their associated structural features $\mathcal{Z}_u$ are *joined* to the query level. Conceptually, query-level structural features $\mathcal{Z}_Q$ act as a proxy to locate the position (i.e., structural information) of node $x \in \mathcal{S}_Q$ in the query-induced subgraph $\mathcal{G}_Q$. Specifically, for a node x in $\mathcal{S}_Q$, its query-level structural features $\mathcal{Z}_{Q,x}$ are obtained by concatenating the node-level structural features $\mathcal{Z}_{u,x}$ for all u in Q as

$$\mathcal{Z}_{Q,x} = ||_{u \in Q} \mathcal{Z}_{u,x} = [\dots \mathcal{Z}_{u,x} \dots] \in \mathbb{R}^{|Q| \times k}, \tag{1}$$

where $||$ denotes concatenation. There will be some $u \in Q$ such that $\mathcal{Z}_{u,x}$ does not exist as node $x \notin S_u$, which is then set to all zeros. For instance, in Fig. 5, node b is in $\mathcal{S}_v$ but not $\mathcal{S}_u$, thus $\mathcal{Z}_{u,b}$ is set to zero. $\mathcal{Z}_Q$ is the collection of $\mathcal{Z}_{Q,x}, \forall x \in \mathcal{S}_Q$. Together, $\mathcal{S}_Q$ and $\mathcal{Z}_Q$ are substituted for the query-induced subgraph $\mathcal{G}_Q$ and later used as input for the neural encoder and to make predictions.

The concatenation in Eq. (1) is equivalent to a full OUTER JOIN operation commonly used in databases to merge tables. Here, to obtain $\mathcal{Z}_{Q,x}$, if the key x matches one of the node indices in $\mathcal{S}_u$, $\forall u \in Q$, then the associated structural features $\mathcal{Z}_{u,x}$ are joined. However, naïvely iterating over all $\mathcal{S}_u, u \in Q$ to retrieve $\mathcal{Z}_{u,x}$ for each node $x \in \mathcal{S}_Q$ is inefficient, as its complexity can be $O(|Q| * |\mathcal{S}_Q|^2)$ for one query Q. This becomes even more challenging to perform these operations for a large number of queries with sets $\mathcal{S}_u, \mathcal{S}_Q$ of varying sizes. To tackle this challenge, we design an efficient sparse arithmetic operator SpJoin to perform joins in parallel on the sparse data structure SpG. This operator reduces the time complexity per query to $O(|Q| * |\mathcal{S}_Q|)$ by taking advantage of the fact that node indices of $\mathcal{S}_u$ stored in SpG are unique and in order. The following uses a query $Q = \{u, v\}$ for demonstration.

Sparse Join Operator The operator SpJoin performs an OUTER JOIN on the node sets SpG$[u, :]$ and SpG$[v, :]$ stored in SpG through

$$\texttt{SpJoin}(\texttt{SpG}[u,:], \texttt{SpG}[v,:]) = \\ \texttt{SpAdd}\,(\texttt{mask}, \texttt{SpG}[u,:]) - 1 \;||\; \texttt{SpAdd}\,(\texttt{mask}, \texttt{SpG}[v,:]) - 1$$

where mask = bool(SpAdd(SpG$[u, :]$, SpG$[v, :]$)). As illustrated in Fig. 5, SpJoin consists of three steps:

(1) The operator leverages sparse arithmetic operations of SciPy [43], with SpAdd performing $X \oplus Y$ to element-wise add the non-zero elements in X and Y, and the bool operator converting SpG objects to binaries, which contain the node indices in the union set $\mathcal{S}_Q$ and are stored in mask.

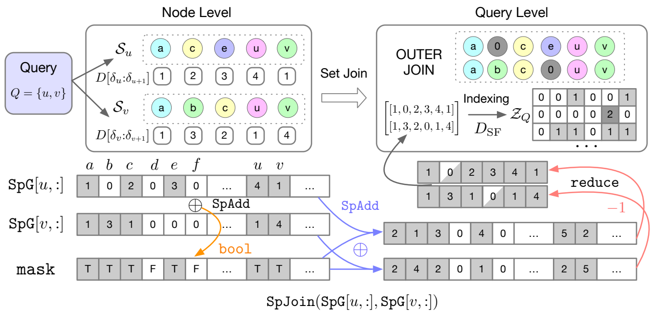


Figure 5: An Illustration of Joining Structural Features from Node-level to Query-level (Eq. (1)) via the SpJoin Operator on Node Sets Stored in SpG Format. Note that SpG objects do not physically contain 0 as above shown in SpG$[u, :]$ and SpG$[v, :]$. Only non-zero elements in grey blocks are stored in SpG and performed arithmetic operations by SpJoin. The half-grey blocks correspond to missing values.

(2) The SpAdd and reduction '-1' are then applied to mask and each SpG$[u, :]$, $u \in Q$, which explicitly adds default missing values (all zeros) of structural features $\mathcal{Z}_{u,x}$ for all x if $x \notin \mathcal{S}_u$ while $x \in \mathcal{S}_v$, for some $v \in Q \backslash \{u\}$.
(3) When the secondary index SFptr D is used, the results of SpJoin are the concatenated pointers of SFptr, which can be used to obtain joined structural features $\mathcal{Z}_Q$ by indexing to gather their values from the array D_{SF}.

Multithreading is adopted to exploit the single program multiple data pattern in these sparse operations. Since the processing time of each query linearly depends on $|\mathcal{S}_Q|$, we further propose to divide the queries of each training batch into groups with nearly balanced sums of $|\mathcal{S}_Q|$'s and have one thread process a group to overcome potential delays caused by uneven workloads.

Comparison with SUREL [51] Despite using the same concatenation to formulate query-level structural features as Eq. (1), SUREL's overall memory consumption is much higher than set-based SUREL+ due to the presence of many duplicated nodes in walks. While the joining for both SUREL and SUREL+ involves sparse operations and runs on the CPU, the memory saving of SUREL+ significantly reduces the workload of data transfer from CPU to GPU. Moreover, SUREL+ requires fewer operations on the GPU side to process transmitted $\mathcal{Z}_Q$'s. These advantages ultimately give SUREL+ much better efficiency and scalability.

3.4 Set Neural Encoders

After joining node sets for each query Q, the resulting $(\mathcal{S}_Q, \mathcal{Z}_Q)$ is utilized as a substitute for the query-induced subgraph $\mathcal{G}_Q$ and then fed into neural encoders for prediction. Next, we introduce the neural encoders supported by SUREL+. The mini-batched training procedure with multiple Q's is summarized in Algorithm 1.

The adopted neural encoders are simple. For each $(\mathcal{S}_Q, \mathcal{Z}_Q)$,

$$h_Q = \mathsf{AGGR}\left(\{enc(\mathcal{Z}_{Q,x}) | x \in \mathcal{S}_Q\}\right) \in \mathbb{R}^d. \tag{2}$$

Here, $enc(\cdot)$ encodes query-level structural features $\mathcal{Z}_{Q,x}$, where a multi-linear perception (MLP) is used. Node attributes, if present, can be appended to the structural features as $\mathcal{Z}_{Q,u}||X_u$. AGGR, which can be any neural encoders applied to sets (e.g. mean/sum/max pooling, set transformers [26], etc.) is used to aggregate the encoded

Algorithm 1: The Training Pipeline of SUREL+

```
Input: Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$, a group of queries
       $\{(Q, y_Q)\}$ for training, batch size $B$, a set SAMPLER, a
       structure ENCODER, and a set AGGR
Output: A neural network for encoding subgraphs $enc(\cdot)$
1 Pre-processing: SAMPLER and ENCODER $\rightarrow (\mathcal{S}_u, \mathcal{Z}_u)$ for all
  $u \in \mathcal{V}$; convert $(\mathcal{S}_u, \mathcal{Z}_u)$'s to SpG objects.
2 for each mini-batch $Q_B = \{..., Q, ...\}$ do
3     Generate negative training queries (if not given) $\{\bar{Q}_i\}$ by
        random sampling and put them into $Q_B$;
4     Perform sparse join operator SpJoin on SpG objects
        $\{(\mathcal{S}_u, \mathcal{Z}_u) | u \in Q\}$ for all queries $Q \in Q_B$ in parallel;
5     Encode the joined proxies of subgraphs $(\mathcal{S}_Q, \mathcal{Z}_Q)$
        according to Eq. (2) where AGGR is specified as the
        input and get the readout $\hat{y}_Q$ in parallel;
6     Backward propagation based on the loss $\mathcal{L}(\hat{y}_Q, y_Q)$.
7 end
```

features. SUREL+ currently supports mean pooling, LSTM, and attention to implement AGGR. Note that the LSTM will perform random permutations of the elements in the set before encoding them as a sequence. Attention will first compute soft attention scores for elements based on the output of $enc(\cdot)$ and then perform attention-score-weighted average pooling. Sec. 4.4 empirically shows the choice of AGGR has a non-trivial effect on prediction performance. Lastly, a fully-connected layer that takes h_Q as input is used to make the final prediction $\hat{y}_Q$. In our experiments, all SGRL tasks can be formulated as binary classification, and Binary Cross Entropy is used as the loss function $\mathcal{L}$.

4 EVALUATION

In this section, we aim to evaluate the following questions:

- Regarding space and time complexity, how much improvement can SUREL+ achieve by adopting sets instead of walks compared to the SOTA SGRL framework SUREL?
- Can SUREL+ provide prediction performance comparable to all baselines that use or do not use SGRL methods?
- How sensitive is SUREL+ to different choices of set samplers, structure encoders, and set neural encoders?
- How do sparse storage SpG and parallelism in SpJoin operator benefit the runtime and scaling performance of SUREL+?

4.1 Experiment Setup

Extensive experiments have been performed to evaluate SUREL+ using homogeneous, heterogeneous, and higher-order homogeneous graphs on three types of tasks, namely, link prediction, relation type prediction, and higher-order pattern prediction. A homogeneous graph is a graph that does not include node/link types, while a heterogeneous graph includes various node/link types. Higher-order graphs are hypergraphs in our setting that contain hyperedges connecting two or more nodes.

Datasets Table 2 summarizes the statistics of the datasets used to benchmark SGRL methods. Five datasets are selected from the Open Graph Benchmark (OGB, [17]) for link and relation type prediction, including social networks of citation - `citation2` and collaboration - `collab`; biological network of protein interaction - `ppa` and vascular - `vessel`; and one heterogeneous academic network `ogb-mag`, which contains node types of paper (P), author (A) and their relations extracted from MAG [46]. The `vessel` dataset has unique significance as a very recent large (>3M nodes), sparse, biological graph extracted from mouse brains [34] for examining GRL in scientific discovery. The structure of vessels illustrates the spatial organization of the brain's microvasculature, which can be used for early detection of neurological disorders, e.g. Alzheimer's and stroke. Two hypergraph datasets collected by [3] are used for higher-order pattern prediction: `DBLP-coauthor` is a temporal hypergraph, where each hyperedge denotes a time-stamped paper connecting all its authors. `tags-math` contains sets of tags applied to questions on the website math.stackexchange.com, represented as hyperedges. For higher-order pattern prediction tasks, the number of hyperedges is the main computation bottleneck, even though the node set is relatively small compared to ordinary graphs. As a hyperedge may connect more than two nodes, it further increases the complexity of the dataset and the need for algorithm scalability.

Table 2: Summary Statistics for Evaluation Datasets.

Dataset	Type	#Nodes	#Edges	Split(%)
`citation2`	Homo./Social.	2,927,963	30,561,187	98/1/1
`collab`	Homo./Social.	235,868	1,285,465	92/4/4
`ppa`	Homo./Bio.	576,289	30,326,273	70/20/10
`vessel`	Homo./Bio.	3,538,495	5,345,897	80/10/10
`ogb-mag`	Hetero.	(P): 736,389 (A): 1,134,649	P-A: 7,145,660 P-P: 5,416,271	99/0.5/0.5
`tags-math`	Higher.	1,629	projected: 91,685 hyperedges: 822,059	60/20/20
`DBLP-coauthor`	Higher.	1,924,991	projected: 7,904,336 hyperedges: 3,700,067	60/20/20

Settings For link prediction, OGB's standard data split is used to isolate validation and test links (queries) from the input graph. For prediction tasks of relation type and higher-order pattern, the same procedure to prepare graph data is adopted as in SUREL [51]: the relations of paper-author (P-A, "written by") and paper-paper (P-P, "cited by") are selected; higher-order queries in hypergraph datasets are node triplets, where the goal is to predict whether it will foster a hyperedge given two of them have observed pairwise connections; to learn the representation on hypergraphs, we project hyperedges into cliques and treat the projection results as ordinary graphs. All experiments are run 10 times independently, and we report the mean performance and standard deviation.

Baselines We consider three classes of baselines. *Canonical GNNs*: GCN [23], GraphSAGE [14], and their variants with the prefix 'H*' that are directly applied for heterogeneous graphs with node types and for hypergraphs through clique expansion. R-GCN [37] that performs relational message passing on heterogeneous graphs; *Scalable GNNs*: GraphSAINT [53] and Cluster-GCN [10]; SGRL models: SEAL [54, 57], GDGNN [25], BUDDY [7], and SUREL [51]. SEAL adopts online subgraph sampling due to the potential huge memory consumption for offline subgraph extraction. Fig. 3 (a) compares the time cost to perform subgraph sampling across different SGRL methods. We adopt the official implementations of all baselines with tuned parameters that match their reported results. BUDDY's source code is not available yet, we adopt their reported results on three OGB datasets from [7].

Table 3: **Prediction Performance for Links, Relation Types and Higher-Order Patterns. The best and the second best are highlighted in bold and underlined accordingly.**

Models	`citation2` MRR (%)	`collab` Hits@50 (%)	`ppa` Hits@100 (%)	`vessel` ROC-AUC
GCN	84.74±0.21	44.75±1.07	18.67±1.32	43.53±9.61
SAGE	82.60±0.36	54.63±1.12	16.55±2.40	49.89±6.78
Cluster-GCN	80.04±0.25	44.02±1.37	3.56±0.40	40.39±2.03
GraphSAINT	79.85±0.40	53.12±0.52	3.83±1.33	47.14±6.83
BUDDY	87.56±0.11	**65.94±0.58**	49.85±0.20	-
GDGNN	86.96±0.28	54.74±0.48	45.92±2.14	75.84±0.08
SEAL	87.67±0.32	63.64±0.71	48.80±3.16	80.50±0.21
SUREL	**89.74±0.18**	63.34±0.52	53.23±1.03	**86.16±0.39**
SUREL+	88.90±0.06	64.10±1.06	**54.32±0.44**	85.73±0.88

Models	`MAG(P-A)` MRR (%)	`MAG(P-P)` MRR (%)	`tags-math` MRR (%)	`DBLP-coauthor` MRR (%)
H*GCN	39.43±0.29	57.43±0.30	51.64±0.27	37.95±2.59
H*SAGE	25.35±1.49	60.54±1.60	54.68±2.03	22.91±0.94
R-GCN	37.10±1.05	56.82±4.71	-	-
SUREL	45.33±2.94	**82.47±0.26**	71.86±2.15	97.66±2.89
SUREL+	**58.81±0.42**	80.45±0.13	**77.73±0.16**	**99.83±0.02**

Hyperparameters By default, SUREL+ uses the walk-based sampler, the structural encoder LP and the better set neural encoder tuned between mean pooling and attention. SUREL+ adopts a 2-layer MLP as $enc(\cdot)$ in Eq. (2) followed by a 2-layer classifier to map set-aggregated representations for final predictions. Default training hyperparameters are: learning rate `lr=1e-3` with the early stopping of 5 epochs, dropout `p=0.1`, Adam [22] as the optimizer. Analysis of parameters M and m to control the walk-based sampler and K to control the metric-based sampler, and selection of structure encoders and set neural encoders are studied in Sec. 4.4.

Evaluation Metrics The evaluation metrics include Hits@P, Mean Reciprocal Rank (MRR), and Area Under Curve (ROC-AUC). Hit@P counts the ratio of positive samples ranked at the top-P place against negative ones. MRR first computes the inverse of the rank of the first correct prediction and then takes the average of obtained reciprocal ranks for a sample of queries. For all datasets adopting MRR, each positive query is paired with 1000 randomly sampled negative test queries, except `tags-math` using 100. ROC-AUC follows the standard definition to measure the model's performance in binary classification.

Environment We use a server with two Intel Xeon Gold 6248R CPUs, 1TB DRAM, and two NVIDIA A100 (40GB) GPUs (only one GPU is used per model). SUREL+ is built on PyTorch 1.12 and PyG 2.2. Set samplers are implemented in C, OpenMP, NumPy, Numba, and `uhash`, integrated into Python training scripts as `SubGAcc` extension; `SpG` is customized based on the CSR format of Scipy.

4.2 Prediction Accuracy Comparison

Table 3 shows the prediction performance of different methods. For these four link prediction benchmarks, SGRL models significantly outperform canonical GNNs and their more scalable variants, especially for two challenging biological datasets `ppa` and `vessel`. Link prediction in biological datasets relies on richer structural information that canonical GNNs have limited expressive power to capture. Within SGRL models, SUREL+ achieves comparable performance to SUREL and outperforms SEAL, which validates the effectiveness of our proposed set-based representations. For predictions of relation type and higher-order pattern, we observe additional performance gains (+2∼13%) from SUREL+ compared to SUREL on three of the four datasets. It is worth noting that there is a large gap between canonical GNNs and SGRL models, particularly in the higher-order case. This demonstrates the inherent limitations of canonical GNNs to make predictions of complex relations involving multiple nodes.

Table 4: **Breakdown of Runtime, Memory Consumption for Different Models on `citation2` and `DBLP-coauthor`. The column Train records the runtime per 10K queries.**

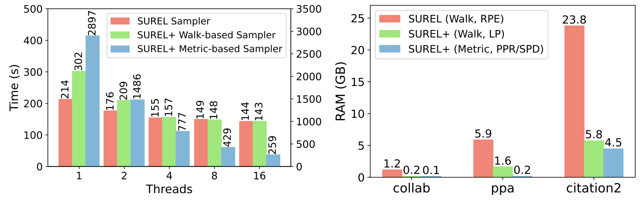
	Models	Runtime (s)			Memory (GB)	
		Prep.	Train	Inf.	RAM	SDRAM
`citataion2`	**GCN**	17	21.74	105	9.3	36.84
	GraphSAINT	151	1.79	107	9.6	9.78
	GDGNN	338	2.26	5,460	40.6	16.96
	SEAL (1-hop)	46	3.52	24,626	35.4	5.71
	SUREL	151	4.14	6,081	25.1	9.68
	SUREL+	130	0.35	1,389	16.7	4.75
`DBLP`	**GCN**	-	0.58	95	8.0	25.80
	SAGE	-	0.32	77	7.5	24.70
	SUREL	11	1.29	949	9.7	7.79
	SUREL+	8	0.24	315	3.8	3.16

SGRL	**Inference Runtime (s)**				
Models	`collab`	`ppa`	`vessel`	`MAG(P-A/P)`	`tag-math`
GDGNN	15	902	84	-	-
SEAL	37	3,988	326	-	-
SUREL	17	1,429	32	1,998/1,924	341
SUREL+	2	201	3	401/168	116

4.3 Efficiency and Scalability Analysis

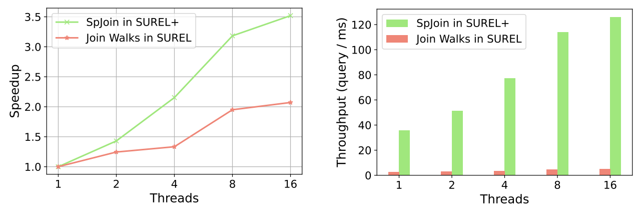
Improved Efficiency in Training and Inference. The upper of Table 4 reports the comparison of runtime and memory across different methods on the two largest datasets `citation2` and `DBLP`. SUREL+ offers a reasonable total runtime compared with canonical GNNs. It shows clear improvement in inference compared to the SOTA SGRL framework SUREL (3-11× speedups) and its predecessor SEAL (∼20× speedups). In terms of memory, SUREL+ achieves comparable and even lower usage than canonical GNNs. Compared to SUREL and other SGRL models, it can save up to half of their RAM, since the set-based representation eliminates duplication of sampled nodes and their associated structural features, which also reduces the memory footprint on the GPU side. The second half of Table 4 summarizes the inference time of SGRL methods on the rest five datasets.

Profiling Different Strategies for Offline Processing Fig. 6a reports the time cost of different samplers with multithreading on `citation2`. Fig. 6b shows memory consumption to store different types of sampled data (walks in SUREL [51] or sets in SUREL+) and their associated structural features (LPs, SPDs, PPR scores). Compared to SUREL sampler[51], the walk-based sampler in SUREL+ is more scalable and only adds an extra minute for encoding and converting data in `SpG` format (slash/dash marked in Fig. 6a), while achieving 6.94×, 3.63× and 4.12× memory savings on three OGB datasets respectively to store sampled sets and their structural features. Those memory savings are more crucial for overall scalability



(a) Runtime **(b) Memory**

Figure 6: Comparison of Runtime, Memory Consumption across Different Offline Processing Strategies (the walk-based sampler: $m = 4, M = 200$, the metric-based sampler: $K = 150$). The areas highlighted break down the total consumption w.r.t. (a) sampling; structure encoding; sparse object construction and (b) structural features, node indices/pointers, and sampled walks (SUREL sampler only).



(a) Speedup **(b) Throughput**

Figure 7: Scaling Performance Comparison of `SpJoin` in SUREL+ (with average set size $|\bar{S_u}| = 351$) and Join Walks in SUREL (with walk size $m = 4, M = 200$) against the Numbers of Threads.

as they reduce the workload of data transfer from CPU to GPU and save many GPU operations to encode the data, which dominates the time cost of the online stage. This leads to the efficiency improvement of SUREL+ in Table 4. In addition, the metric-based sampler that adopts PPR scores has better scaling performance when using more threads. When adopting PPR scores or SPDs as structural features, SUREL+ further reduces the memory cost, though later we see that they often do slight harm to the prediction performance.

Note that in the above comparison of memory cost, compressed structural features are adopted both in SUREL (locally) and SUREL+ (globally), i.e., the secondary index based on `SFptr` D and D_{SF}, which achieve compression of 493×, 11318×, 19527× on three datasets listed in Fig. 6b, when one adopts LPs as structural features.

Scaling Analysis for `SpJoin` Fig. 7 shows the speedups and throughput of the sparse join operator `SpJoin` via multithreading, where the join operation in SUREL to construct query-level structural features for nodes on walks [51] is used as a comparison. SUREL uses a hash-based search for walk joins, which has unfavorable memory access patterns and suffers from workload imbalance due to inconsistent searching times in hash tables among different threads. SUREL+ earns more benefits from multithreading, thanks to the use of `SpJoin` and batch-wise load balancing.

4.4 Comparison between Different Set Samplers, Structural Features and Set Neural Encoders

SUREL+ is a modularized framework that supports different set samplers (walk- and metric-based), structural features (LP, SPD, PPR), and set neural encoders `AGGR` (mean pooling, LSTM, attention).

Table 5 shows the prediction performance and inference runtime by adopting different combinations of structure encoders and set neural encoders. Landing probabilities (LPs) as structural features

Table 5: Prediction Performance and Inference Runtime of SUREL+ with Different Combinations of Structure Features (LP, SPD, PPR) and Set Neural Encoders (Mean, LSTM, Attn.). The best and the second best are highlighted in bold and underlined accordingly.

Dataset	PPR+Mean	SPD+Mean	LP+Mean	LP+LSTM	LP+Attention
`citation2`	78.59±0.38	87.99±1.07	<u>88.55±0.15</u>	88.46±0.34	**88.90±0.06**
	<u>965s</u>	**847s**	1389s	3678s	2171s
`collab`	47.15±0.21	62.11±0.13	**64.10±1.06**	61.31±1.37	<u>62.85±1.19</u>
	1.4s	<u>1.7s</u>	2.0s	3.5s	2.3s
`ppa`	13.28±1.20	41.06±1.70	46.41±1.65	**54.45±1.35**	<u>54.32±0.44</u>
	63s	<u>126s</u>	165s	322s	201s

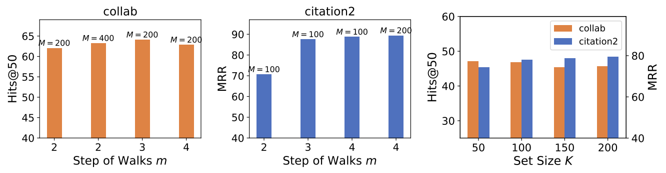


Figure 8: Hyperparameter Analysis of Set Samplers. Walk-based: the number M and the step m of walks, LPs as structural features; Metric-based: the set size K, PPR scores as structural features.

perform the best on all three OGB datasets while being the slowest for inference. By recording the landing probabilities over different steps of walks, LPs provide structural information in finer granularity than both SPDs and PPR scores which are just scalars. Also, it might be due to the adopted link prediction task, which favors more local information held by LPs and SPDs than global information carried by PPR scores. The authors conjecture that other tasks that rely on more global information may favor PPR scores. In comparison, there is not a set neural encoder as an always winner. Attention seems to perform the best on average while slower than mean pooling. LSTM is the slowest. On the two social networks (`citation2` and `collab`), mean pooling can provide comparable prediction results with much fewer parameters. However, prediction on the biological network (`ppa`) requires more expressive and complicated encoders, where LSTM and attention are favored as they can model more complex interactions between nodes in $\mathcal{S}_Q$.

Fig. 8 compares prediction results by using different hyperparameters m, M, and K of set samplers, which heavily affects the coverage of the neighborhoods of sampled node sets. The performance consistently increases if a larger M is used in the walk-based sampler, but it is not guaranteed for a larger m. Better coverage with a larger K is generally beneficial for the metric-based sampler over `citation2` but not for `collab`, which is due to different characteristics of these two datasets and is also observed by [51]. In general, small sampling parameters m $(2 \sim 4)$, M $(100 \sim 400)$ and K $(50 \sim 200)$ can yield satisfactory performance. By adjusting them, we can achieve a trade-off between accuracy and scalability.

5 CONCLUSION

In this work, we propose a novel framework SUREL+ for scalable subgraph-based graph representation learning. SUREL+ avoids the costly procedure of extracting subgraphs for specific queries by sampling node sets whose join can be used as a proxy of query-induced subgraphs for prediction. SUREL+ benefits from the reusability of presampled node sets across different queries. Compared to the SOTA framework SUREL based on walk sampling, the set-based representation of SUREL+ substantially reduces memory and time consumption by avoiding heavy node duplicates in sampled walks. To handle irregularly sized node sets, SUREL+ designs a dedicated sparse storage SpG and a sparse join operator SpJoin providing memory-efficient storage and fast access. In addition, SUREL+ adopts a modular design that enables users to flexibly choose different set samplers, structure encoders, and set neural encoders to remedy the loss of structural information after the reduction from subgraphs to sets in their own SGRL tasks. Extensive experiments on three types of prediction tasks over seven real-world graph benchmarks show that SUREL+ achieves significant improvements in scalability, memory efficiency, and prediction accuracy compared to current SGRL methods and canonical GNNs.

ACKNOWLEDGMENTS

The authors would like to thank Rongzhe Wei and Yanbang Wang for their helpful discussions and valuable feedback. H. Yin and P. Li are supported by the 2021 JPMorgan Faculty Award and the National Science Foundation (NSF) award OAC-2117997.

REFERENCES

[1] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. 2020. Subgraph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 8017–8029.

[2] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 475–486.

[3] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230.

[4] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2464–2473.

[5] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[6] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. 2021. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3747–3756.

[7] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. 2023. Graph Neural Networks for Link Prediction with Subgraph Sketching. In *International Conference on Learning Representations*.

[8] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*.

[9] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. 2020. Can graph neural networks count substructures? *Advances in Neural Information Processing Systems* 33 (2020), 10383–10395.

[10] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 257–266.

[11] DGL. 2022. 6.7 Using GPU for Neighborhood Sampling — DGL 0.9.1post1 documentation. https://docs.dgl.ai/guide/minibatch-gpu-sampling.html

[12] Fabrizio Frasca, Beatrice Bevilacqua, Michael M Bronstein, and Haggai Maron. 2022. Understanding and Extending Subgraph GNNs by Rethinking Their Symmetries. *Advances in Neural Information Processing Systems* 35 (2022).

[13] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. 2020. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*. PMLR, 3419–3430.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* 30 (2017), 1025–1035.

[15] William L Hamilton. 2020. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14, 3 (2020), 1–159.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40, 3 (2017), 52–74.

[17] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems* 33 (2020), 22118–22133.

[18] Kexin Huang and Marinka Zitnik. 2020. Graph meta learning via local subgraphs. *Advances in Neural Information Processing Systems* 33 (2020), 5862–5874.

[19] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*. 271–279.

[20] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.

[21] Tim Kaler, Nickolas Stathas, Anne Ouyang, Alexandros-Stavros Iliopoulos, Tao Schardl, Charles E Leiserson, and Jie Chen. 2022. Accelerating training and inference of graph neural networks with fast sampling and pipelining. *Proceedings of Machine Learning and Systems* 4 (2022), 172–189.

[22] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[23] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[24] Daphne Koller, Nir Friedman, Sašo Džeroski, Charles Sutton, Andrew McCallum, Avi Pfeffer, Pieter Abbeel, Ming-Fai Wong, Chris Meek, Jennifer Neville, et al. 2007. *Introduction to statistical relational learning*. MIT press.

[25] Lecheng Kong, Yixin Chen, and Muhan Zhang. 2022. Geodesic Graph Neural Network for Efficient Graph Representation Learning. *Advances in Neural Information Processing Systems* 35 (2022).

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

[27] Pan Li, I Chien, and Olgica Milenkovic. 2019. Optimizing generalized pagerank methods for seed-expansion community detection. *Advances in Neural Information Processing Systems* 32 (2019), 11710–11721.

[28] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning. *Advances in Neural Information Processing Systems* 33 (2020), 4465–4478.

[29] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. 2020. Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1959–1969.

[30] Yunyu Liu, Jianzhu Ma, and Pan Li. 2022. Neural Predicting Higher-Order Patterns in Temporal Networks. In *Proceedings of the Web Conference 2022*. ACM, 1340–1351.

[31] Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, Jure Leskovec, et al. 2020. Neural Subgraph Matching. *arXiv preprint arXiv:2007.03092* (2020).

[32] Yuhong Luo and Pan Li. 2022. Neighborhood-aware Scalable Temporal Network Representation Learning. *Learning on Graphs Conference* (2022).

[33] Changping Meng, S Chandra Mouli, Bruno Ribeiro, and Jennifer Neville. 2018. Subgraph pattern neural networks for high-order graph evolution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[34] Johannes C Paetzold, Julian McGinnis, Suprosanna Shit, Ivan Ezhov, Paul Büschl, Chinmay Prabhakar, Anjany Sekuboyina, Mihail Todorov, Georgios Kaissis, Ali Ertürk, et al. 2021. Whole Brain Vessel Graphs: A Dataset and Benchmark for Graph Learning and Neuroscience. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[35] Jingshu Peng, Zhao Chen, Yingxia Shao, Yanyan Shen, Lei Chen, and Jiannong Cao. 2022. Sancus: staleness-aware communication-avoiding full-graph decentralized training in large-scale graph neural networks. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1937–1950.

[36] PyG. 2022. Accelerating PyG on NVIDIA GPUs. https://www.pyg.org//ns-newsarticle-accelerating-pyg-on-nvidia-gpus

[37] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.

[38] Yili Shen, Jiaxu Yan, Cheng-Wei Ju, Jun Yi, Zhou Lin, and Hui Guan. 2022. Improving Subgraph Representation Learning via Multi-View Augmentation. *arXiv preprint arXiv:2205.13038* (2022).

[39] Balasubramaniam Srinivasan and Bruno Ribeiro. 2020. On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations*.

[40] Balasubramaniam Srinivasan, Da Zheng, and George Karypis. 2021. Learning over Families of Sets-Hypergraph Representation Learning for Higher Order Tasks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 756–764.

[41] Komal Teru, Etienne Denis, and Will Hamilton. 2020. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*. PMLR, 9448–9457.

[42] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations.*

[43] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272.

[44] Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. BNS-GCN: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling. *Proceedings of Machine Learning and Systems* 4, 673–693.

[45] Cheng Wan, Youjie Li, Cameron R Wolfe, Anastasios Kyrillidis, Nam Sung Kim, and Yingyan Lin. 2022. Pipegcn: Efficient full-graph training of graph convolutional networks with pipelined feature communication. In *International Conference on Learning Representations.*

[46] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies* 1, 1 (2020), 396–413.

[47] Xiyuan Wang and Muhan Zhang. 2021. GLASS: GNN with Labeling Tricks for Subgraph Representation Learning. In *International Conference on Learning Representations.*

[48] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *International Conference on Learning Representations.*

[49] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Xiaojie Guo. 2022. Graph neural networks: foundation, frontiers and applications. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 4840–4841.

[50] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations.*

[51] Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, and Pan Li. 2022. Algorithm and System Co-design for Efficient Subgraph-based Graph Representation Learning. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2788–2796.

[52] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 19665–19679.

[53] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations.*

[54] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems* 31 (2018), 5165–5175.

[55] Muhan Zhang and Yixin Chen. 2020. Inductive Matrix Completion Based on Graph Neural Networks. In *International Conference on Learning Representations.*

[56] Muhan Zhang and Pan Li. 2021. Nested graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 15734–15747.

[57] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2021. Labeling Trick: A Theory of Using Graph Neural Networks for Multi-Node Representation Learning. *Advances in Neural Information Processing Systems* 34 (2021), 9061–9073.

[58] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. 2022. TGL: A General Framework for Temporal GNN Training on Billion-Scale Graphs. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1572–1580.

A NOTATIONS

Frequently used symbols are summarized in Table 6.

Table 6: Summary of Frequently Used Notations.

Symbol	Meaning
Q	a query (set of nodes), i.e. $Q = \{u, v, w\}$
$\mathcal{Q}$	a collection of queries, i.e. $Q \in \mathcal{Q}$
$\mathcal{G}_u$	a subgraph induced by node u
$\mathcal{G}_Q$	a subgraph induced by query Q
$\mathcal{S}_u$	a unique node set sampled from the neighborhood of the seed node u
$\mathcal{Z}_{u,x}$	structural features of node x regarding the seed node u (all zeros if $x \notin \mathcal{S}_u$)
$\mathcal{Z}_u$	collection of structural features for all nodes in $\mathcal{S}_u$ as $\mathcal{Z}_u = \{\mathcal{Z}_{u,x} \| x \in \mathcal{S}_u\}$
$\|\|$	the concatenation that joins node-level structural features, i.e. join $\mathcal{Z}_{\cdot,x}$ for a query $Q = \{u, v, w\}$ as $[\mathcal{Z}_{u,x}, \mathcal{Z}_{v,x}, \mathcal{Z}_{w,x}]$.
$\mathcal{Z}_{Q,x}$	query-level structural features for node x regarding the query Q, $\mathcal{Z}_{Q,x} = \|\|_{u \in Q} \mathcal{Z}_{u,x}$

B MORE DETAILS

B.1 Model Design

Benefits of Subgraph-based Graph Representation Learning Firstly, subgraph-based representation is versatile for different types of tasks, especially when queries of certain tasks involving multiple nodes and relations, e.g. existence of a link, property of a network motif, development of higher-order patterns; while canonical GNNs are limited to handle such polyadic dynamics via single-node representations [39, 47] (also refers to the example in Fig. 1). Secondly, subgraph-based models are more expressive by pairing with structural features to obtain structural node representations [5, 28, 39, 47]. However, canonical GNNs are incapable of capturing intra-distance information, which is critical to predict over a set of nodes and to distinguish nodes with structural symmetry. Lastly, subgraph-based methods decouple the model depth from the receptive field because extracted subgraphs are localized: when adding more layers for non-linearity, it does not contaminate embedding with irrelevant nodes or get over-smooth as canonical GNNs do. This results in a more robust representation and is particularly beneficial for modeling relations beyond singleton.

B.2 Datasets

The full statistics of benchmark datasets are summarized in Table 7. OGB datasets[1] are selected to benchmark our proposed framework and other baselines, due to large-scale graphs (millions of nodes/edges) for real-world applications (i.e. networks of academic, biological networks) it contains, and standard, open-sourced evaluation metrics and tools it provides. Note that, `vessel` is a newly added benchmark of a biological graph, with $> 3M$ nodes and sparse vessel structures extracted from the whole mouse brain [34], where nodes represent bifurcation points, and edges represent the vessels. Each node is associated with features of its physical location in the coordinate space (x, y, z). The introduction of `vessel` provides a unique opportunity to examine graph representation learning (GRL) approaches in the neuroscience domain, especially in scaling subgraph-based GRL (SGRL) methods to handle sparse and spatial graphs with millions of nodes and edges for scientific discovery.

B.3 Baselines

For link prediction and relation type prediction, baseline models are selected based on their scalability and prediction performance from the current OGB leaderboard [2]. All models listed on the leaderboard

[1] https://ogb.stanford.edu/docs/dataset_overview/

[2] https://ogb.stanford.edu/docs/leader_linkprop/

Table 7: Summary Statistics and Experimental Setup for Evaluation Datasets.

Dataset	Type	#Nodes	#Edges	Avg. Node Deg.	Density	Split Ratio	Split Type	Metric
citation2	Homo.	2,927,963	30,561,187	20.7	0.00036%	98/1/1	Time	MRR
collab	Homo.	235,868	1,285,465	8.2	0.0046%	92/4/4	Time	Hits@50
ppa	Homo. / Bio.	576,289	30,326,273	73.7	0.018%	70/20/10	Throughput	Hits@100
vessel	Homo. / Bio.	3,538,495	5,345,897	3.02	0.000085%	80/10/10	Random	AUC-ROC
ogb-mag	Hetero.	Paper(P): 736,389 Author(A): 1,134,649	P-A: 7,145,660 P-P: 5,416,271	21.7	N/A	99/0.5/0.5	Time	MRR
tags-math	Higher.	1,629	91,685 (projected) 822,059 (hyperedges)	N/A	N/A	60/20/20	Time	MRR
DBLP-coauthor	Higher.	1,924,991	7,904,336 (projected) 3,700,067 (hyperedges)	N/A	N/A	60/20/20	Time	MRR

Table 8: Hyperparameters Used for Benchmarking SUREL+.

Dataset	#steps m	#walks M	#neg samples k	Structural Feature	Set Neural Encoder
citation2	4	100	10	LP	Mean
collab	3	200	10	LP	Mean
ppa	4	200	20	LP	Attn.
vessel	2	50	5	LP	Mean
MAG (P-A)	3	200	10	LP	Mean
MAG (P-P)	4	100	10	LP	Mean
tags-math	4	200	10	LP	Mean
DBLP-coauthor	3	100	10	LP	Mean

have publicly accessible code attached with a technical report. We adopt their published numbers if available on the leaderboard with additional verification. For the rest baselines, we benchmark these models using their official implementations and tuning parameters as listed below.

- **GCN family**: a graph auto-encoder model that uses graph convolution layers to learn node representations, including GCN [23], GraphSAGE [14], and their more scalable variants by employing graph subsampling, such as Cluster-GCN [10], GraphSAINT [53].
- **R-GCN**[3] [37]: a relational GCN that models heterogeneous graphs with node/link types.
- **SEAL**[4] [54]: apply GCN on exact query-induced whole subgraphs with double radius node labeling to obtain subgraph-level readout for link prediction. SEAL shows great empirical performance on multiple graph machine learning benchmarks and promotes the deployment of subgraph-based models for scientific discovery. The implementation we tested is specially optimized for OGB datasets provided in [57].
- **SUREL**[5] [51]: a walk-based computation framework to accelerate subgraph-based models, where subgraphs are decomposed to reusable walks and then online joined to represent the query-induced subgraph for prediction. By adopting walk-based representation, SUREL achieves state-of-the-art scalability and prediction accuracy on SGRL tasks.
- **GDGNN**[6] [25]: a model aggregates node representation generated by GNNs along geodesic paths between nodes in a query to speed up prediction for SGRL tasks.

All canonical GNN baselines[7] come with three `GCNConv`/`SAGEConv` layers of 256 hidden dimensions, and a tuned dropout ratio in $\{0, 0.5\}$ for full-batch training. Canonical GNN models aggregate all node embeddings involved in a query as the representations of links/hyperedges, which are later fed into an MLP classifier for final prediction. In addition, all GNN models need to use full training data (edges/triplets) to generate robust node representations. The hypergraph datasets do not come with raw node features, thus GNN baselines here use random features as input for training along with other model parameters. R-GCN uses `RGCNConv` layers that support message passing with multiple relation types between different types of nodes, where the edge types (relations) are used as input beside node features.

Subgraph-based models only use partial edges/triplets from the training set. 1-hop enclosing subgraphs are extracted online during the training and inference of SEAL. Then, it applies three GCN layers of hidden dimension 32 plus a sort pooling and several 1D convolution layers to generate a readout of the target subgraph for prediction. SUREL consists of a 2-layer MLP for the embedding of relative position encoding (RPE) and a 2-layer RNN to encode joined walks attached with RPEs to represent query-induced subgraphs. The hidden dimension of both networks is set to 64. The obtained embeddings of joined walks are aggregated and fed into a 2-layer MLP classifier to make predictions. GDGNN employs `GINLayer` as

[3]https://github.com/pyg-team/pytorch_geometric/blob/master/examples
[4]https://github.com/facebookresearch/SEAL_OGB
[5]https://github.com/Graph-COM/SUREL

[6]https://github.com/woodcutter1998/gdgnn
[7]https://github.com/snap-stanford/ogb/tree/master/examples/linkproppred

its backbone to obtain node embeddings. The horizontal geodesic representation is used for predictions, which finds the shortest path between two nodes in a query and aggregates node representations generated by GNNs along the found geodesic path. The max search distance for geodesic is the same as the number of GNN layers. For `collab`, `ppa`, `citation2` and `vessel`, the threshold of distance is set to 4, 4, 3, and 2, respectively. The hidden dimension of all fully-connected layers is set to 32.

C ARCHITECTURE AND HYPERPARAMETER

SUREL+ uses a 2-layer MLP with ReLU activation for the encoding of structural features and supports three set neural encoders including mean pooling, LSTM, and attention. LSTM interprets elements to be aggregated in a set as a sequence [14]; attention first calculates soft attention scores for elements in a set and then performs attention-score-weighted average pooling. The hidden dimension of all parameterized layers is set to 96. Lastly, hidden representations of joined node sets are fed into a 2-layer MLP classifier for final predictions.

For the implementation of set samplers, the walk-based sampler builds on the sampling function from `SubGAcc`[8] library developed by the authors, which also provides the support for efficient structural feature hashing and index remapping. The metric-based sampler is adopted from fast PPR approximation in [4].

For link and relation prediction, we follow the inductive setting: only partial samples will be used for training. Over the training graph, we randomly select 5% links as positive training queries, each paired with k-many negative samples ($k = 10$ by default). We mask these links and use the remaining 95% links to compute structural features for each node in the split training set via structure encoder. For `vessel`, as the input graph is very sparse, we first sort the nodes in training set by their degree and then randomly pick 5% nodes to obtain edges of their 3-hop induced subgraphs for training and the rest reserved for constructing structural features. For higher-order pattern prediction, we use the given graph before timestamp t to sample node sets and encode their structural features. The model parameters are optimized by triplets provided in the training set. No node features are used in SUREL+, except for `vessel` where normalized physical locations of each node are attached after its structural features.

The results reported in Table 3 and the profiling of SUREL+ in Table 4 are obtained through the combination of hyperparameters listed in Table 8. The dropout rate on `vessel` is set to `p=0.2`. For the results of using structural features PPR and SPD in Table 5, the metric-based sampler is used. Its sampling size K is set to 50, 50 and 150 for `collab`, `ppa`, `citation2`, respectively. For the results of using structural features LP, the walk-based sampler is used, and its sampling parameters are listed in Table 8. The rest hyperparameters remain the same as reported in Sec. 4.1.

[8]https://github.com/VeritasYin/subg_acc