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

Haoteng Yin[†], Muhan Zhang[‡], Jianguo Wang[†], Pan Li[†]
[†]Department of Computer Science, Purdue University [‡] Institute for Artificial Intelligence, Peking University
[†]{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 expressive power and generalization ability. Most of previous SGRL models suffer from the computational issue of the costly extraction of a subgraph for each training or testing query. Recently, SUREL has been proposed as a new framework to accelerate SGRL. SUREL samples random walks offline and join these walks online as subgraphs to make predictions. Due to the reusability of these walks across different queries, SUREL achieves state-of-the-art performances in both scalability and prediction accuracy. However, SUREL still suffers from a severe computational overhead induced by node redundancy in the sampled walks. In this work, we propose a novel framework SUREL+ that upgrades SUREL by utilizing node sets instead of walks to represent subgraphs. Node sets by definition avoid node duplication. To address the irregularity induced by node sets, we design dedicated sparse data structures and operations that allow fast accessing node sets and joining them in parallel batches. SUREL+ is also modularized to support multiple set samplers, encoding techniques and structural features to complement the loss of structural information due to the reduction from walks to sets. Extensive experiments have been performed to validate SUREL+ on the prediction tasks of links, relation types, and higher-order patterns. SUREL+ achieves 3-5× speedup of SUREL while having comparable or even better prediction performance; Compared to other SGRL frameworks, SUREL+ achieves 13-38 × speedup with significant prediction accuracy improvement.

PVLDB Reference Format:
Haoteng Yin[†], Muhan Zhang[‡], Jianguo Wang[†], Pan Li[†]. SUREL+: Moving from Walks to Sets for Scalable Subgraph-based Graph Representation Learning [Scalable Data Science]. 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.

1 INTRODUCTION

Graphs are widely used to model interactions in natural sciences and relationships in social life [19, 23]. Real-world graph-structured data are highly irregular and often in a large scale. To solve inference tasks on graphs, graph representation learning (GRL) that studies learning quantitative representations of graph-structured data has attracted much attention [14, 15, 48]. Recently, subgraph-based GRL (SGRL) has become one important research direction for researchers who are studying GRL algorithms and systems, as SGRL approaches have 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, named a queried node set, SGRL approaches such as SEAL [53, 56], GraIL [40] and SubGNN [1] first extract a subgraph around the queried node set, and then encode the extracted subgraph to make a prediction. Extensive works have shown that SGRL models are more robust [51] and more expressive [5, 11]; while canonical graph neural networks (GNNs) including GCN [22] and GraphSAGE [13] generally fail to make accurate predictions, due to their limited expressive power [8, 12, 56], incapability of capturing intra-node distance information [27, 38], and improper entanglement of the sizes of receptive fields and the model depth [17, 50, 51]. An example in Fig. 1 illustrates how SGRL works for link prediction, and shows its advantage over canonical GNNs that generate and aggregate node representations to predict links. Here, canonical GNNs will map nodes in structural symmetry into the same representation and cause the ambiguity issue [49, 56]. So far, the advantages of SGRL models have been proved in many applications, such as link and relation prediction [40, 53, 56], high-order pattern prediction [29, 32], temporal network modeling [47], recommender systems [54], anomaly detection [1, 6], graph meta-learning [17], subgraph matching [28, 30], and molecular/protein study in life science [37, 46].

Albeit with multiple advantages of their algorithms, current SGRL models face two major challenges on the computation side that hinder their deployment in practice: (1) **Query Dependency**. A subgraph needs to be extracted for each queried node set, which is not reusable among different queries and cannot be preprocessed if the query is unknown ahead; (2) **Irregularity**. Extracted subgraphs are irregular, causing performance degradation in batch processing and load balancing. As Fig. 3 (a) shows, subgraph extraction in SEAL [53, 55] is very costly. This motivates the recent studies on dedicated hardware acceleration for subgraph extraction [10, 35]. However, how to improve the scalability and computation of SGRL methods is still largely undeveloped.

SUREL [50] is the state-of-the-art (SOTA) work which applies algorithm and system co-design to achieve SGRL. It adopts reusable node-level walks to represent query-specific subgraphs. Specifically, SUREL regards each node in the graph as a seed and offline samples a collection of random walks that start from the seed. In the online phase, given a queried node set, SUREL joins and encodes the walks

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

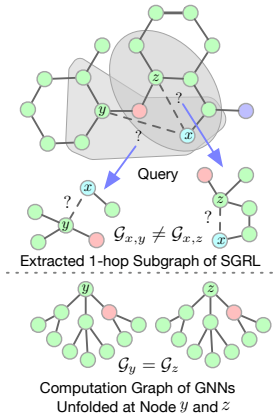


Figure 1: GNNs cannot correctly predict whether x is more likely linked with y or z, because y and z have the same node representations. However, the representations based on one-hop neighbors are more expressive to distinguish the two node pairs.

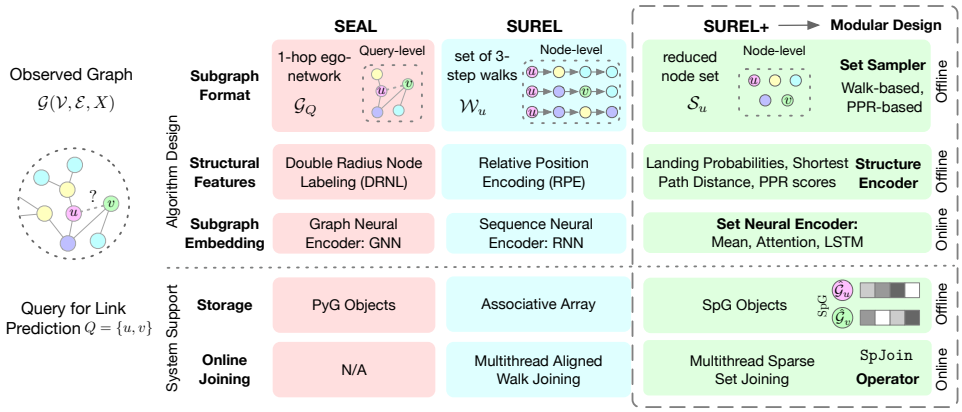


Figure 2: Overview of SUREL+: Side-by-Side Comparison with Previous SGRL Frameworks via an Example for Link Prediction. SUREL+ sample node sets while SEAL [53, 56] extracts the whole subgraph for each query and SUREL [50] sample walks. To serve node set-based representations, SUREL+ designs a new algorithm with dedicated system design. SUREL+ supports various types of set samplers, structure encoders, and set neural encoders to complement the loss of structural information by using 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 batches via the sparse arithmetic operator SpJoin.

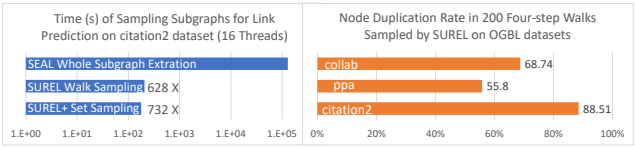


Figure 3: (a) Subgraph extraction in SEAL [53, 56] has much higher complexity compared to other samplers with simplification and suffers from irregularity and "neighborhood explosion". (b) Breaking subgraphs into reusable walks reduces complexity but also faces the issue of high node duplication.

for all nodes in the query to make prediction. The joint walks here essentially act as a proxy of the subgraph for this queried node set. To compensate for the loss of structural information caused by using walks to represent the subgraph, structural features termed relative position encoding to locate each node in the subgraph are adopted and pre-computed offline, and are fed into neural networks (NNs) to make final predictions. In SUREL, walks from one seed node will be reused in multiple queries as long as multiple queries involve this node. SUREL significantly improves the scalability of SGRL based on this walk-sharing mechanism, the regularity of sampled walks, and its dedicated system design that supports highly parallel walk sampling and online joining. However, SUREL still faces many computational issues caused by the inherent shortcomings of walk-based representations, i.e., super high node duplication in the sampled walks (over 55%, see Fig. 3 (b)). This raises issues including (1) **extra space cost** of storing walks on the CPU side, (2) **high workload** of data transfer from CPU to GPU, and (3) **extra time consumption** of the redundant operations in walk joining and in subsequent NN-based encoding 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, which evolves from using walks to using node sets to represent subgraphs since the latter removes node duplication. However, this new node set-based idea introduces many algorithm and system challenges. Regarding the algorithm, the reduction from the whole subgraph to walks and further to node sets studied in this work loses much structural information. How to compensate such a loss and to have an algorithm without performance decay is a big challenge for algorithm design. From the system side, the walks used in SUREL [50] can be stored and processed in a regular format by controlling with the length and the number of the sampled walks. However, node sets are irregular. How to efficiently store and process irregular node sets poses many challenges to system design. In particular, how to coordinate the designs of both sides is the main challenge of this work to resolve.

SUREL+ addresses the above challenges throughout its pipeline. In the first step, for each node in the graph, SUREL+ offline extracts a subset of unique nodes from the neighborhood and associates the sampled nodes with structural features. SUREL+ supports various types of *set samplers* and *structure encoders* to construct structural features to compensate the loss of structural information. Specifically, set samplers leverage different graph metrics to measure the importance of a node to decide the sampling rules. Structure encoders use landing probability of random walks (LP) [26], shortest path distance (SPD), and personalized PageRank (PPR) scores [18], extensively covering the structural features adopted by previous SGRL methods [27, 40, 46, 50, 53, 56]. SUREL+ also designs a dedicated sparse data structure SpG to store the sampled node sets that are memory efficient and allow fast access. In the second step, given a query, SUREL+ will online join the sampled node sets around the seed nodes in the query and the associated structural features to represent the subgraph for this query to make a prediction. SUREL+ adopts an SpJoin operator that can perform the join operation of node sets in parallel batches with provable speed improvement. Moreover, SUREL+ supports multiple *set neural encoders* including multi-linear perception (MLP) + mean pooling, set attention [41] and LSTM [13], which can capture different levels of interactions between structural features to guarantee good final predictions.

Overall, our contributions can be summarized as follows:

- Algorithm: SUREL+ is a novel SGRL framework (*open source*), which adopts reusable sets of unique nodes and associates them with flexible and various types of structural features, whose online joins represent subgraphs for SGRL queries. Though using sets with much memory saving, SUREL+ achieves no decay in the prediction performance compared to the SOTA baselines.
- System: SUREL+ adopts dedicated sparse data structure `SpG` and arithmetic operation `SpJoin` to support efficient storing and processing node sets, which achieves much better scalability and efficiency than previous SGRL methods.
- We have done extensive experiments to demonstrate the advantages of SUREL+ through link/relation/motif prediction tasks over 7 real-world graphs with millions of nodes/edges. SUREL+ achieves 3-5 × speedup over the SOTA SGRL method SUREL with comparable or even better prediction accuracy. SUREL+ also achieves 13-38 × speedup with significant prediction accuracy improvement over other SGRL methods.

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 tasks. We denote the query-induced subgraph induced by Q as $\mathcal{G}_Q$ and the node-induced subgraph as $\mathcal{G}_u$, where induced subgraphs are typically within small number of hops.

Next, we formally define subgraph-based GRL.

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 the query Q corresponds to a pair of nodes. It can be further generalized to predict links with types over heterogeneous graphs [40], or to predict vascular access [33] and chemical bond [19] for domain-specific graphs. Tasks beyond pairwise relations are named *higher-order pattern prediction*, where the set Q consists of three or more nodes. In this work, we consider that given partially observed relations among queried nodes in Q, whether these nodes will establish certain full higher-order relation of interest [29, 39].

Review of Current SGRL Pipelines SGRL frameworks have 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 typically coupled with queries in classical SGRL models, e.g., SEAL [53, 56]. However, such coupling is expensive and the underlying computation cannot be shared among queries, which motivated later SGRL methods to decouple them. SUREL [50] replaces explicit subgraph extraction with the online joins of multiple walks that can be pre-sampled offline. Structural features RPEs defined based on sampled walks are used. Those walks and structural features can be shared to assemble subgraphs for multiple queries, which improves the reusability and model scalability. NNs are applied afterwards to encode and aggregate these walks to make predictions.

2.2 Related Works

Scalable GNN Design. GNNs are currently the most widely used tool for GRL, though they suffer from some issues when being directly used as SGRL models. Most current studies on scaling GNNs focus on improving graph subsampling and mini-batch training techniques [9, 52]. However, graph subsampling for GNNs is fundamentally different from the step of subgraph extraction used in SGRL: the former aims to address GPU memory overflow of full-graph training, while the latter is part of the SGRL algorithms where sampling subgraphs around queried nodes or node sets of interest defines the features to make prediction over. With different goals, the techniques of scaling GNNs are inapplicable to SGRL models. Several distributed GNN systems are purposed to deploy on industrial-level graphs, by hiding or reducing costs in communication and synchronization through specialized system design such as pipelining [44], partitioned parallelism [43] and update with staleness [34]. Unfortunately, they are not directly useful to address the challenges of SGRL models, such as subgraph extration.
Scalable SGRL Design. Recent system works for scaling SGRL models focuses on efficient subgraph extraction. These include PPR-based [4, 51] and random walk-based [50] subgraph samplers, sampling node neighborhood via CUDA kernel in DGL [10], via tensor operation in PyG [35], through performance-engineered sampler in SALIENT [20], and parallel sampling for temporal graphs [57]. Several frameworks achieve much higher throughput by customizing data structures to better support subgraph operations, including associative array in SUREL [50], temporal-CSR in TGL [57] and GPU-orientated dictionary in NAT [31]. For scalable modeling designs, ShaDow [51] decouples the depth and scope of GNNs by restricting receptive fields to localized subgraphs. To avoid the high complexity of structural feature construction, GDGNN [24] employs representations along geodesic paths to make prediction for a query with multiple nodes. ELPH [7] uses subgraph sketches that avoid explicitly construct subgraphs, while being only applicable to link prediction. All these works only partially address the scalability issues or can be applied to very specific tasks (e.g., link prediction) in SGRL compared to SUREL+.

3 THE FRAMEWORK OF SUREL+

In this section, we introduce SUREL+. The main idea of SUREL+ is to offline pre-sample node sets from the neighborhoods of nodes in the graph, where the node sets can be joined online as a proxy to represent query-induced subgraphs and can be shared across different queries. Beyond reusability, such set-based representations also address the computation and memory issues in SUREL [50] caused by node duplication in the walks adopted in SUREL. SUREL+ has modular designs that support different set samplers, structure encoders to construct various types of structural features, and multiple set neural encoders, which can be chosen flexibly and compensate for the loss of structural information. SUREL+ also adopts dedicated sparse data structure `SpG` and arithmetic operator

SpJoin to store and perform online set joins in batches, respectively. A direct comparison between SUREL+ and current SGRL approaches is shown in Fig. 2. In the following subsections, we introduce the specifics of these modules.

3.1 Set Samplers and Structure Encoders

SUREL+ adopts set samplers to sample a node set from the neighborhood of each node in the graph and calls structure encoders to construct structural features. Both of the operations are performed offline. The former is mainly for computational benefits while the latter is to compensate for the loss of structural information due to the reduction from subgraphs (adopted by SEAL [53, 56]) or walks (adopted by SUREL [50]) to sets. Conceptually, SUREL+ represents the node-induced subgraph $\mathcal{G}_u$ via a combination of (a) a unique node set $\mathcal{S}_u$ that contains sampled nodes from the neighborhood of node u and (b) the associated structural features $\mathcal{Z}_u$ that reflect the position in $\mathcal{G}_u$ of each sampled node in $\mathcal{S}_u$.

Set Samplers Our adopted set samplers are in two types. The first type named *Walk-based Sampler* is to sample short-step random walks and then remove duplicated nodes on sampled walks. The second type named *Metric-based Sampler* is based on other more principled graph metrics that measure the proximity between a node and the seed node, such as PPR scores [18] and short path distances (SPDs). Specifically, the walk-based sampler runs M-many m-step random walks that start from each node u in parallel on graph $\mathcal{G}$ ($\mathcal{G}$ stored in the compressed sparse row (CSR) format), and only puts unique nodes on these walks into the set $\mathcal{S}_u$. The metric-based sampler, taking PPR [4] as an example, first runs the push-flow algorithm [2] to obtain an approximation of PPR vector for each seed node u. Then, nodes with top-K PPR scores are selected in $\mathcal{S}_u$. Mathematically, PPR scores are convergent landing probabilities of seeded random walks that reach infinite steps. So, the two samplers essentially complement each other by leveraging more local or more global structures of the graph. Here, we have hyper-parameters M, m to control random walks and K to control PPR, which are set as some contants in practice. The complexity of the above offline sampling procedures are $O(|\mathcal{V}|)$.

Structure Encoders Structure Encoders are to construct structural features $\mathcal{Z}_{u,x} \in \mathbb{R}^k$ for each node x in the sampled node set $\mathcal{S}_u$. Structural features can be conceptually understood as to take the seed node u as the anchor and then locate node x within the neighborhood of node u. Such structural features are provably crucial for inference tasks involving multiple nodes [56]. One possible choice is random-walk landing probabilities (LPs) [26, 27, 50]: 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 from u divided by the total number of walks. By definition, LPs can be constructed along with random walks sampling process. Another choice is the SPD between x and u [27, 53, 56]. Also, PPR scores [18] are also useful structural features, which can be computed along with the metric-based sampler running. Later, we denote the group of structural features for all nodes in $\mathcal{S}_u$ as $\mathcal{Z}_u = \{\mathcal{Z}_{u,x} | x \in \mathcal{S}_u\}$.

3.2 Set-based Storage - SpG

Node-set-based representations have benefits in their reusability and duplication removal. However, they pose challenges to their storage and fast access due to their irregularity. Note that these node sets need to be accessed frequently in the latter online stage to make further predictions. To address these challenges, SUREL+ reorganizes the storage of node sets with their structural features in a dedicated compressed sparse row (CSR) format, termed SpG. As Fig. 4 shows, one node set $\mathcal{S}_u$ and its structural features $\mathcal{Z}_u$ are stored in a row of SpG as SpG$[u, :]$. Multiple node sets and their structural features are coalesced in three consecutive arrays:

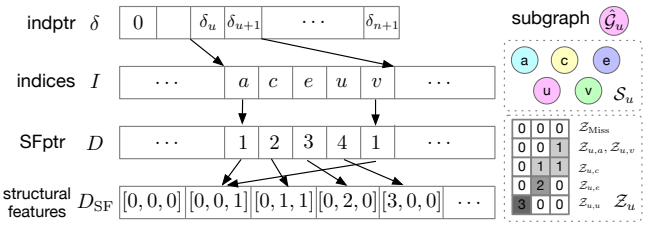


Figure 4: Node set $\mathcal{S}_u$ and its associated structural features $\mathcal{Z}_u$ Stored in SpG. Here, D_{SF} stores the counts of nodes that appear at different steps in the sampled walks, which can be further normalized to structural features LPs and is shown here as an example.

- `indptr` $\delta \in \mathbb{Z}^{n+1}$, an array points to where each stored node set (row) begins. It records cumulative sum of the size of $\mathcal{S}_u$ for each node $u \in \mathcal{V}$, i.e., $\delta_{u+1} = \delta_u + |\mathcal{S}_u|$. The total number of nodes from all node sets explicitly-stored in SpG is δ_{n+1};
- `indices` $I \in \mathbb{Z}^{\delta_{n+1}}$, a coalesced array of node indices in $\mathcal{S}_u, \forall u \in \mathcal{V}$. The consecutive segment $I[\delta_u : \delta_{u+1}]$ corresponds to the node indices in $\mathcal{S}_u$. These indices are stored in a sorted order, which is super useful to accelerate the join operations latter studied in Sec. 3.3.
- `SFptr` $D \in \mathbb{R}^{\delta_{n+1}}$, a pointer array contains the indices of the structural features $\mathcal{Z}_u$ that are stored in array D_{SF}. D_{SF} eliminates duplicate structural features by hashing and usually resides in GPU memory. Note that when LPs/SPDs are used as structural features, there could be a lot of duplications and this secondary index can further improve memory efficiency. When PPR scores are used, their values do not have many duplications.

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

Comparison with Other Methods Table 1 summarizes the comparison. δ_{n+1} is about $0.2mM|\mathcal{V}|$ by adopting the walk-based sampler (sampling M-many m-step walks) or $K|\mathcal{V}|$ by adopting the metric-based sampler (sampling top-K PPR scores), which are substantially lower than $O(S|\mathcal{E}|)$ that SEAL uses and about one fifth of $mM|\mathcal{V}|$ used by SUREL. For hosting structural features, SUREL+ adopts the secondary index `SFptr` D and only keeps unique structural features in D_{SF}, which further squeezes memory consumption. As c is typically independent of $|\mathcal{V}|$ in practice, it makes SUREL+ with SpG beneficial to deal with the case with large graphs.

3.3 Joining Node Sets via Sparse Operations

The goal of joining node sets is to construct a proxy of the query-induced subgraph based on the sampled node sets to make prediction for a query. For a given query Q, we online *join* relevant node

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 the dimension 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 [22]	SEAL [53, 56]	SUREL [50]	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	$O(d\lvert\mathcal{V}\rvert)$	$O(kS\lvert\mathcal{E}\rvert)$	$O(mMk\lvert\mathcal{V}\rvert)$	$O(\delta_{n+1} + c * k)$

sets into $\mathcal{S}_Q = \bigcup_{u\in Q} \mathcal{S}_u$ and their associated structral features $\mathcal{Z}_u$ to query-level structural features $\mathcal{Z}_Q$. Conceptually, these query-level structural features work as a proxy to locate the position (i.e., structural information) of a node $x \in \mathcal{S}_Q$ in the query-induced subgraph $\mathcal{G}_Q$. Specifically, for a node x in $\mathcal{S}_Q$, we use $\mathcal{Z}_{Q,x}$ to denote its query-level structural features, which is obtained by concatenating of 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 the concatenation. There will be some $u \in Q$ such that $\mathcal{Z}_{u,x}$ does not exist as $x \notin \mathcal{S}_u$, which is then set to all zeros. $\mathcal{Z}_Q$ works as the collection of $\mathcal{Z}_{Q,x}$, $x \in \mathcal{S}_Q$. $\mathcal{S}_Q$ and $\mathcal{Z}_Q$ together gives a proxy of the query-induced subgraph $\mathcal{G}_Q$, which are later used as the input of the neural encoder and to make a prediction.

Note that $\mathcal{S}_u$, $\mathcal{S}_Q$ as sets are irregular. To perform the concatenation in Eq. (1), naïvely searching each node $x \in \mathcal{S}_Q$ over all $\mathcal{S}_u, u \in Q$ is inefficient, whose complexity for each query Q could be as much as $O(|Q| * |\mathcal{S}_Q|^2)$. Also, as there are often a large number of queries, the irregularity of sets makes it difficult to perform these operations in parallel for batches of queries. We observe that the concatenation here is essentially the full OUTER JOIN, a common way to merge tables used in database literatures. Here, it is performed to join structural features $\mathcal{Z}_u$ over all $\mathcal{S}_u, u \in Q$. As the node indices in $\mathcal{S}_u$ are unique and in a sorted order stored in the SpG format, we may reduce the complexity for each query to $O(|Q| * |\mathcal{S}_Q|)$. Our implementation is based on an efficient sparse arithmetic operator SpJoin defined as follows.

Sparse Join Operation SpJoin The following uses the query $Q = \{u, v\}$ for demonstration. An example is illustrated in Fig. 5. SpJoin performs OUTER JOIN on node sets SpG$[u, :]$ and SpG$[v, :]$ stored in SpG format 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$[v, :]$)). All leverage sparse arithmetic operations of SciPy [42]. SpAdd performs $X \oplus Y$ that do element-wise addition for non-zero elements in X and Y. bool operator converts SpG objects to binaries, which include the node indices in the union set $\mathcal{S}_Q$ and are stored in mask. Next, SpAdd and reduction '-1' are further 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 $v \in Q\backslash\{u\}$. As the secondary index SFptr D is used, the results of SpJoin are the concatenated SFptrs. To obtain joined structural features $\mathcal{Z}_Q$, indexing can be called to gather their values from the array D_{SF}. Here, we adopt multithreading to exploit the single program multiple data (SPMD) pattern in this sparse operation. As the processing

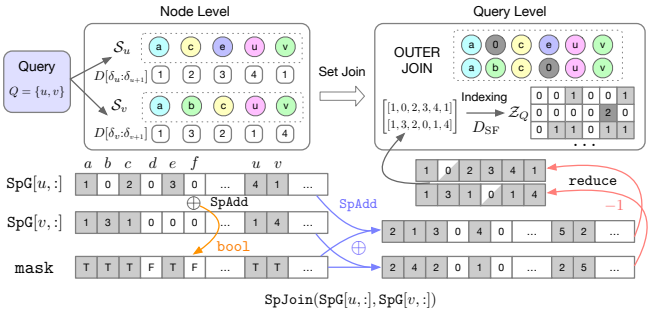


Figure 5: An Illustration of Joining Structural Features from Node-level to Query-level (Eq. (1)) via the SpJoin Operator on Node Sets stored in SpG Format. Note that SpG objects do not physically contain 0 as above shown in SpG$[u, :]$ and SpG$[v, :]$. Only non-zero elements in grey blocks are stored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

time of each query linearly depends on $|\mathcal{S}_Q|$, we also propose to partition queries in each training batch into almost groups with almost balanced sums of $|\mathcal{S}_Q|$'s and let each thread process each group to overcome the potential delay caused by unbalancing.

Comparison with SUREL [50] Although SUREL also adopts concatenation as Eq. (1) to formulate query-level structural features, as there are a lot of duplicated nodes in walks, the overall memory consumption of SUREL will be much higher than SUREL+ that adopts sets. As the join operations in both SUREL and SUREL+ involve sparse operations and run on the CPU side, memory saving of SUREL+ will significantly reduce the workload of data transition from CPU to GPU. Also, SUREL+ will save much more operations on the GPU side to process $\mathcal{Z}_Q$'s transmitted to GPU. 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 $(\mathcal{S}_Q, \mathcal{Z}_Q)$ works as a proxy of the query-induced subgraph $\mathcal{G}_Q$ and will go through neural encoders and make 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 adopt neural encoders are simple. For each $(\mathcal{S}_Q, \mathcal{Z}_Q)$,

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

Here, $enc(\cdot)$ encodes query-level structural features $\mathcal{Z}_{Q,x}$, where we use an MLP. Node attributes, if exist, can be appended after structural features as $\mathcal{Z}_{Q,u}||X_u$. AGGR can be any neural encoders applied to sets, e.g. mean/sum/max pooling, set transformers [25], and so on. SUREL+ currently supports mean pooling, LSTM, and attention to implement AGGR. Note that LSTM here will perform random permutations of the elements in the set before encoding them as a sequence. Attention will first calculate soft attention scores for the 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 will affect the performance non-trivially. Lastly, a fully connected layer that takes h_Q as input is used for the final prediction $\hat{y}_u$. In our experiments, all SGRL tasks can be formulated as binary classification, and thus we employ Binary Cross Entropy as the loss function $\mathcal{L}$.

Algorithm 1: The Training Pipeline of SUREL+

Input: Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$, a set 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 $enc(\cdot)$

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.

for *each mini-batch* $\mathcal{Q}_B = \{..., Q, ...\}$ **do**

  Generate negative training queries (if not given) $\{\bar{Q}_i\}$ by random sampling and put them into $\mathcal{Q}_B$;

  Perform sparse join operator SpJoin on SpG objects $\{(\mathcal{S}_u, \mathcal{Z}_u) | u \in Q\}$ for all queries $Q \in \mathcal{Q}_B$ (batchwise parallelization) ;

  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 $\hat{y}_Q$ (batchwise parallelization);

  Backward propagation based on the loss $\mathcal{L}(\hat{y}_Q, y_Q)$.

end

4 EVALUATION

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

- Regarding time and space complexity, how much improvement can SUREL+ achieve compared to the SOTA SGRL framework SUREL by adopting sets instead of walks?
- 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 operation SpJoin of SUREL+ benefit runtime and scaling performance?

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 which contain hyperedges connecting two or more nodes.

Datasets Table 2 summarizes the statistics of the datasets used to benchmark SGRL approaches. Five datasets are selected from Open Graph Benchmark [16] 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 [45]. The vessel dataset has unique significance as a very recent large ($> 3M$ nodes), sparse, biological graph extracted from mouse brains [33] to examine GRL in scientific discovery. The structure of vessels illustrates the spatial organization of the brain's microvasculature, which can be used to detect neurological diseases in the early stage, 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 that connect all its authors. tags-math contains sets of tags applied to questions on the website math.stackexchange.com, which are denoted as hyperedges. For higher-order pattern prediction tasks, the number of hyperedges is the main computation bottleneck, even if the node set is relatively small compared to standard graphs. As a hyperedge may connect more than two nodes, it further increases the dataset complexity 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, the standard data split of OGB is used to isolate the validation and test links (queries) from the input graphs. For prediction tasks of relation type and higher-order pattern, the same procedure to prepare graph data is adopted as in SUREL [50]: the relations of paper-author (P-A) and paper-paper (P-P) are selected; the 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 connection; 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 deviations.

Baselines We consider three classes of baselines. *Canonical GNNs*: GCN [22], GraphSAGE [13], 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 [36] that performs relational message passing on heterogeneous graphs; *Scalable GNNs*: GraphSAINT [52] and Cluster-GCN [9]; SGRL models: SEAL [53, 56], GDGNN [24], SUREL [50]. SEAL adopts online subgraph sampling due to the potential huge memory consumption for offline subgraph extraction. Fig. 3 (a) compares the time costs to perform subgraph sampling across different SGRL methods. We adopt the official implementations of all baselines with tuned parameters that match their reported results.

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 early stopping of 5-epoch patience, dropout p=0.1, Adam [21] as the optimizer. Analysis of 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 investigated in Sec. 4.4.

Evaluation Metrics The evaluation metrics include Hits@P, Mean Reciprocal Rank (MRR) and Area Under Curve (ROC-AUC).

Table 3: Prediction Performance for Links, Relation Types and Higher-Order Patterns. The best and the second best are highlighted in bold and with underlines 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
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**	**54.02±0.87**	85.73±0.88

Models	MAG(P-A)	MAG(P-P)	tags-math	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**

Hit@P counts the ratio of positive samples ranked at the top-P place against negative ones. MRR firstly computes inverse of the rank of 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 which uses 100. ROC-AUC follows the standard definition to measure model's performance to perform binary classification.

Environments 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 1.10 and PyG 2.1. Set samplers are implemented in C, OpenMP, NumPy, Numba and uhash, integrated into the Python training script through C extension; SpG is customized based on the CSR format of Scipy.

4.2 Prediction Accuracy Comparison

Table 3 shows prediction performances of different methods. For these four link prediction benchmarks, the SGRL model significantly outperforms canonical GNNs and their more scalable variants, especially for the challenging biological datasets (ppa, 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 mostly outperforms SEAL, which validates the effectiveness of our proposed set-based representations. For relation type prediction and higher-order pattern prediction, 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.

4.3 Efficiency and Scalability Analysis

Improved Efficiency in Training and Inference. The upper of Table 4 reports the runtime, memory comparison across different methods on the two largest benchmarks citation2 and DBLP.

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

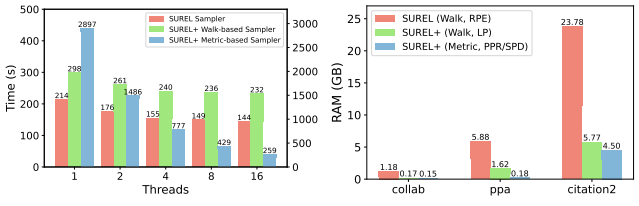
	Models	Runtime (s)			Memory (GB)	
		Prep.	Train	Inf.	RAM	SDRAM
citation2	**GCN**	17	21.74	105	9.3	36.84
	GraphSAINT	151	1.79	107	9.6	9.78
	SEAL (1-hop)	46	3.52	24,626	35.4	5.71
	GDGNN	338	2.26	5,460	40.6	16.96
	SUREL	151	4.14	6,081	25.1	9.68
	SUREL+	258	0.35	1,647	22.4	4.25
DBLP	**GCN**	-	0.58	95	8.0	25.80
	SAGE	-	0.32	77	7.5	24.70
	SUREL	11	1.29	949	9.7	7.79
	SUREL+	28	0.24	337	3.5	5.46

SGRL	Inference Runtime (s)				
Models	collab	ppa	vessel	MAG	tag-math
SEAL	37	6,851	1,076	-	-
GDGNN	15	902	84	-	-
SUREL	21	2,761	175	353	123
SUREL+	8	551	26	77	62

SUREL+ offers a reasonable total runtime on these benchmarks compared with canonical GNNs. It shows clear improvement on inference compared to the SOTA SGRL framework SUREL (3 to 5× speedup) and its predecessor SEAL (13-38× speedup). In terms of memory cost, 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 design eliminates the most duplications of sampled nodes and their structural features, which also reduces the memory cost on the GPU side. We further summarize inference time of SGRL methods on the rest benchmarks in the second half of Table 4. Next, we investigate three computation intensive modules of SUREL+.

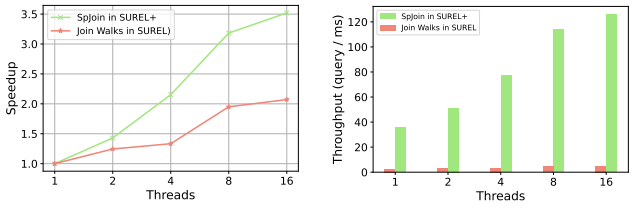
Profiling Different Strategies for Offline Processing Fig. 6a reports the time costs of different samplers with multithreading on citation2. Fig. 6b shows memory consumption to store different types of sampled data (walks in SUREL [50] or sets in SUREL+) and their associated structural features (LPs, PPR scores, SPDs). Compared to walk-based sampler in SUREL [50], the walk-based sampler in SUREL+ adds extra ∼ 30% one-time cost of constructing the storage in SpG format (slash/dash marked), while saving a lot of memory (6.94×, 3.63× and 4.12× memory savings on three OGB datasets respectively) to store the sampled sets and structural features. Those memory savings are more crucial for overall scalability as they reduce the workload of data transition from CPU to GPU and save many GPU operations to encode the data, which dominate 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 has even 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 on memory consumption, both walks in SUREL and sets in SUREL+ adopt compressed structural features, i.e., the secondary indexing based on SFptr D and



(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; structural encoding; sparse object construction and (b) structural features, node indices and pointers, 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.

D_{SF}, which achieve compression of 121×, 2842×, 4891× on three datasets, when one adopts LPs as structural features.

Parallelism Analysis for `SpJoin` Fig. 7 shows the speedups and throughput of the sparse join operation `SpJoin` via multithreading, where the join operation in SUREL to construct query-level structural features for nodes on walks [50] is used as comparison. SUREL uses hash-based search for concatenation, which has unfavorable memory access patterns, and suffers from workload imbalance due to inconsistent searching times of hash tables among different threads. SUREL+ earns more benefits from multithreading, thanks to the use of `SpJoin` and load balancing.

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

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

Table 5 shows the prediction performances and inference runtimes by adopting different combinations of structure encoders and set neural encoders. LPs as structural features perform the best on all three datasets while are the slowest for inference. LP by recording the landing probabilities over different steps of walks provides structural information in a 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 in average while slower than Mean. LSTM is the slowest. On the two social networks (`citation2` and `collab`), mean pooling

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

Dataset	PPR+Mean	SPD+Mean	LP+Mean	LP+LSTM	LP+Attention
citation2	78.59±0.38	87.99±1.07	88.55±0.15	88.46±0.34	**88.90±0.06**
	965s	**847s**	1647s	4008s	2245s
collab	47.15±0.21	62.11±0.13	**64.10±1.06**	61.31±1.37	62.85±1.19
	1.5s	3.4s	8.0s	13.4s	13.2s
ppa	13.28±1.20	41.06±1.70	46.41±1.65	**54.45±1.35**	54.02±0.87
	94s	275s	483s	1020s	551s

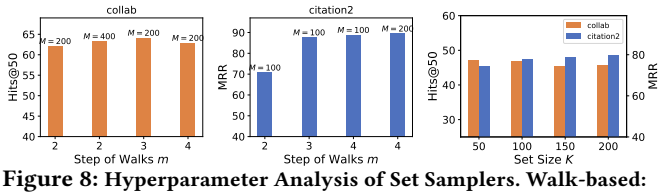


Figure 8: Hyperparameter Analysis of Set Samplers. Walk-based: the number M and the step m of walks, LPs as structural features; Metric-based: the set size K, PPR scores as structural features.

can provide comparable predicition results while with much less parameters. However, prediction on the biological network (ppa) requires more expressive and complicated encoders, where LSTM and attention are favored as they can model 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 the 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. A better coverage with a larger K is generally beneficial for the metric-based sampler over `citation2` but is not over `collab`, which is due to different characteristics of these two datasets and is also observed by [50]. In general, some small sampling parameters $m(2 \sim 4), M(100 \sim 400)$ and $K(50 \sim 200)$ ensure adequate 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 (SGRL). SUREL+ avoids the costly procedure of query-specific subgraph extraction while using sampled node sets whose join can be used as proxies of the subgraphs to make prediction. SUREL+ benefits from the reusability of these node sets across different queries. Compared to the SOTA framework SUREL based on sampling walks, SUREL+ with these node sets substantially reduces memory/time consumption by avoiding having heavy node duplication as that in walks. SUREL+ designs dedicated sparse storage `SpG` and sparse join operation `SpJoin` to handle the irregularity of node sets. SUREL+ also adopts a modular design, where users may flexibly choose among different set samplers, structure encoders and set neural encoders to compensate the loss of structural information caused by set-based representations for their own SGRL tasks. Extensive experiments based on three types of prediction tasks over seven real-world graph benchmarks have demonstrated significant improvements of SUREL+ in scalability, memory efficiency and prediction accuracy compared to current SGRL methods and canonical GNNs.

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. 2022. Graph Neural Networks for Link Prediction with Subgraph Sketching. *arXiv preprint arXiv:2209.15486* (2022).

[8] 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.

[9] 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.

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

[11] 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).

[12] 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.

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

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

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

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

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

[23] 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.

[24] 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).

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

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

[32] 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.

[33] 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*.

[34] 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.

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

[36] 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.

[37] 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).

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

[39] 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.

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

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

[42] 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.

[43] 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.

[44] 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*.

[45] 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.

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

[47] 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*.

[48] 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.

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

[50] 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.

[51] 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.

[52] 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*.

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

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

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

[56] 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.

[57] 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.

Table 6: Summary of Frequently Used Notations.

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

A NOTATIONS

Frequently used symbols are summarized in Table 6.

B MORE DETAILS

B.1 Model Design

Benefits of Subgraph-based GRL Firstly, subgraph representation is versatile for different types of tasks, especially when queries of certain tasks involve with 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 [38, 46], and 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, 27, 46]. However, canonical GNNs are incapable of capturing intra-distance information, which is critical to predict over a set of nodes and distinguish nodes with structural symmetry. Lastly, subgraph methods decouple the model depth from the receptive field because the extracted subgraphs are localized, which does not contaminate with irrelevant nodes and get over-smooth as canonical GNNs do when adding more layers for non-linearity. This results in a more robust representation, and is particularly beneficial for modeling relations beyond singleton.

B.2 Datasets

The full statistic of benchmark datasets is summarized in Table 7. OGB datasets[1] are selected to benchmark our proposed framework and other baselines, due to large-scale graphs (million of nodes/edges) for real-world applications (i.e. network of academic, biological network) it contains and standard, open-sourced evaluation metrics and tools it provides. Note that, `vessel` is newly added benchmark of biological graph, with $> 3M$ nodes and sparse vessel structures extracted from the whole mouse brain [33], where node represent bifurcation points, and edges represent the vessels. Each node is associated with features including its physical location in the coordinate space (x, y, z) and the anatomical location in reference to the Allen brain atlas (one-hot encoded). The introduction of `vessel` provides a unique opportunity to examine GRL for scientific discovery, especially in scaling SGRL approaches to handle such sparse and spatial graphs with millions of nodes and edges.

B.3 Baselines

For link prediction and relation type prediction, baseline models are select based on their scalability and prediction performance from the current OGB leaderboard [2]. All models listed on the leaderboard have public 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 using graph convolution layers to learning node representations, including GCN [22], GraphSAGE [13], and their more scalable variants by employing graph subsampling, such as Cluster-GCN [9], GraphSAINT [52].
- **R-GCN**[3] [36]: a relational GCN that models heterogeneous graphs with node/link types.
- **SEAL**[4] [53]: apply GCN on exact query-induced subgraphs with double radius node labelling to obtain subgraph-level readout for link prediction. SEAL shows great empirical performance on multiple graph machine learning benchmarks and promotes the deployment of SGRL models for scientific discovery. The implementation we tested is specially optimized for OGB datasets provided in [56].
- **SUREL**[5][50]: a walk-based computation framework for subgraph-based GRL, 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 the SOTA scalability and prediction accuracy on SGRL tasks.
- **GDGNN**[6] [24]: a recent SGRL model that aggregates the representation generated by GNNs along the geodesic path between nodes in a query for prediction.

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 the 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, where 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.

SGRL-based models only use partial edges/triplets from the training set. 1-hop enclosing subgraphs are extracted online during the

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

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

[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

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	Attention
vessel	2	400	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

training and inference of SEAL. Then, it applies three GCN layers of hidden dimension 32 plus a sortpooling and several 1D convolution layers to generate readout of the target subgraphs for prediction. SUREL consists of a 2-layer MLP for the embedding of node RPEs and a 2-layer RNN to encode concatenated walks through online joining to represent query-induced subgraphs. The hidden dimension of both networks is set to 64. The obtained hidden representations of joined walks are aggregated and fed into an 2-layer MLP classifier to make predictions. GDGNN employs `GINLayer` as 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 the node representation 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. 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-based, and attention based. LSTM-based, which performs LSTM-style aggregation and interprets elements to be aggregated as a sequence [13]; attention-based, which first calculates soft attention scores like GAT [41], and then performs weighted average pooling. The hidden dimension of all parameterized layers is set to 96. Lastly, hidden representations of joined node sets are fed into an 2-layer classifier to make final predictions.

For the implementation of set samplers, the walk-based sampler builds on the random walk sampler from `surel_gacc` library provided by [50], and library `fastremap`[8] to support fast remapping of secondary index after hashing structural features. The metric-based sampler is adopted from fast PPR approximation in [4].

For link and relation prediction, we follow the inductive setting that 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 ($k = 50$ by default). We remove these links and use the remaining 95% links to compute structural features for each node in the split training set via structure encoder. For higher-order pattern prediction, we use the given graph before timestamp t to sample node sets and their structural features, and then optimize model parameters by triplets provided in the training set. No node features are used in benchmarking SUREL+, except for `vessel` where anatomical position features are attached after 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. 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/seung-lab/fastremap