Algorithm and System Co-design for Efficient Subgraph-based Graph Representation Learning

Haoteng Yin[†], Muhan Zhang[‡], Yanbang Wang[§], Jianguo Wang[†], Pan Li[†]
[†]Department of Computer Science, Purdue University [‡] Institute for Artificial Intelligence, Peking University
[§]Department of Computer Science, Cornell University
[†]{yinht, csjgwang, panli}@purdue.edu [‡] muhan@pku.edu.cn [§] ywangdr@cs.cornell.edu

ABSTRACT

Subgraph-based graph representation learning (SGRL) has been recently proposed to deal with some fundamental challenges encountered by canonical graph neural networks (GNNs), and has demonstrated advantages in many important data science applications such as link, relation and motif prediction. However, current SGRL approaches suffer from scalability issues since they require extracting subgraphs for each training or test query. Recent solutions that scale up canonical GNNs may not apply to SGRL. Here, we propose a novel framework SUREL for scalable SGRL by co-designing the learning algorithm and its system support. SUREL adopts walk-based decomposition of subgraphs and reuses the walks to form subgraphs, which substantially reduces the redundancy of subgraph extraction and supports parallel computation. Experiments over six homogeneous, heterogeneous and higher-order graphs with millions of nodes and edges demonstrate the effectiveness and scalability of SUREL. In particular, compared to SGRL baselines, SUREL achieves 10× speed-up with comparable or even better prediction performance; while compared to canonical GNNs, SUREL achieves 50% prediction accuracy improvement.

PVLDB Reference Format:
Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, Pan Li. Algorithm and System Co-design for Efficient Subgraph-based Graph Representation Learning. PVLDB, 15(11): 2788 - 2796, 2022.
doi:10.14778/3551793.3551831

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

1 INTRODUCTION

Graph-structured data is prevalent to model relations and interactions between elements in real-world applications [19]. Graph representation learning (GRL) aims to learn representations of graph-structured data and has recently become a hot research topic [11]. Previous works on GRL focus on either model design or system design while very few works jointly consider them. Works on model design tend to propose more expressive, generalizable and robust GRL models while paying less attention to their deployment [27, 34]. Hence, many theoretically powerful models can hardly apply to large real-world graphs. On the other hand, research on system design focuses on system-level techniques for better model development, such as graph partitioning [6], sub-sampling [12, 45] and pipelining [16, 33, 46, 50]. However, they only consider basic GRL models, in particular graph neural network (GNN) models, yet often overlook their modeling limitations to solve practical GRL tasks.

Canonical GNNs [12, 18] share a common framework: each node is associated with a vector representation that gets iteratively updated by aggregating the representations from its neighboring nodes via graph convolution layers. The final prediction is made by combining the representations of nodes of interest. Although recent successes in system research have greatly pumped up the efficiency [8, 37], the GNN framework intrinsically suffers from three modeling limitations. First, information may be over-squashed into a single node representation that results in subpar performance when multiple tasks are associated, e.g. to predict multiple relations or links attached to the same node [1, 7]. Second, canonical GNNs cannot capture intra-node distance information due to limited expressive power [20, 35], and thus fail to make predictions over a set of nodes (See Fig. 1a), such as substructure counting [2, 5] and higher-order pattern prediction [31, 49]. Third, the depth of GNNs is entangled with the range of the receptive field. For more non-linearity, using deeper GNNs comes with a larger but possibly unnecessary receptive field, which poses the risk of contaminating the representations with irrelevant information [14, 44].

Recently, subgraph-based GRL (SGRL) has emerged as a new trend and has shown superior performance in tasks such as link prediction [47, 49], relation prediction [32], higher-order pattern prediction [20, 24], temporal network modeling [39], recommender systems [48], graph meta-learning [14], and subgraph matching [23, 25] and prediction [38]. Different from canonical GNNs, SGRL extracts a subgraph patch for each training and test query and learns the representation of the extracted patch for final prediction (See Fig. 1b). For example, SEAL [47, 49] learns the representation of a subgraph around a given node pair to predict the link between them. This framework fundamentally overcomes the above three limitations. First, subgraph extraction allows decoupling the contributions made by a node to different queries, which prevents information over-squashing. Second, subgraph patches can be paired with distance-related features that favor prediction over a set of nodes [20, 49]. Third, subgraph extraction disentangles model depth and range of receptive field, which allows learning a rather non-linear model with only relevant local subgraphs as input.

Despite their importance, the SGRL framework has not received as much attention as the canonical GNN framework in the system

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551831

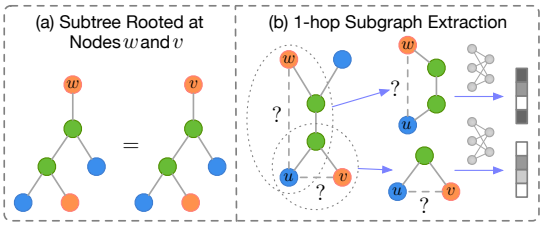


Figure 1: A Toy Example of SGRL: the task is to predict whether uw or uv is more likely to form a link. Ideally, if this comes from a social network, uv is more likely linked because they share a common neighbor. However, canonical GNNs cannot tell such difference since w and v share the same subtree structures resulting in the same representation [40]. SGRL solves this problem by extracting a subgraph patch around each queried node pair. Prediction based on the subgraph representation provides much better performance than canonical GNNs [47, 49].

research community. The underlying challenge comes from the subgraph extraction step in SGRL, which can be rather irregular and time-consuming. Specifically, SGRL requires to materialize a subgraph patch for each query during training and inference. Previous works of SGRL typically extract subgraphs offline for all such queries [24, 47], but it is not scalable for large graphs due to extensive memory need. Meanwhile, the online extraction [44] is not an option as it requires considerable processing time. The irregularity of subgraphs further makes it difficult to efficiently handle the extraction process in both cases.

Here, we aim to fill the gap by designing a novel computational framework SUREL, to support SGRL over large graphs. SUREL consists of a new system-friendly learning algorithm for SGRL and a scalable system to support this algorithm. The crucial design of SUREL is to reduce the overhead caused by the online subgraph extraction, which all current SGRL approaches suffer from.

The key idea behind SUREL is to break (and down-sample) subgraphs into random walks of regular size that can be easily sampled and, more importantly, reused among different queries. To compensate for the missing structural information after subgraph decomposition, we introduce relative position encoding (RPE), an intra-node distance feature that records the position of each node in the sampled subgraph. Specifically, for each node u in the network, SUREL collects a certain number of random walk starting from u. Each node appearing in these walks uses its landing counts at each step as the RPE vector. Overall, the set of collected walks paired with RPEs can be viewed as a subgraph patch centered at u. The complexity of the above process is linear with the number of nodes, and can be done in parallel and offline. For training and inference, given a queried node set Q, SUREL first groups the sampled walks originated from all nodes in Q. Then, it implicitly joins the subgraph patches centered at each node in Q by combining their node-level RPEs into a query-level RPE for each node associated in the grouped walks, which can also be executed in full parallel. Finally, SUREL uses neural networks to learn the representation of the joined set of walks attached with query-level RPEs for final prediction. Since these walks are regular, the training process can be done quickly by GPU. The system architecture of SUREL is illustrated in Fig. 2.

Our contributions can be summarized as follows: (1) **A Novel System-Friendly Algorithm.** We propose the first scalable algorithm for SGRL tasks by adopting a novel walk-based computation framework. This framework uses regular data structures and allows extreme system acceleration. (2) **Dedicated System Support (Open-source).** We design SUREL to support the proposed algorithm. It can rapidly sample walks, encode positional features, and join them to represent multiple subgraphs in parallel. SUREL adopts many system optimization techniques including parallelization, memory management, load balancing, etc. (3) **High Performance and Efficiency.** We evaluate SUREL on link/relation/motif three prediction tasks over 6 real-world graphs of millions of nodes/edges. SUREL significantly outperform the current SGRL approaches, and executes 10× faster in training and testing. Meanwhile, benefiting from the SGRL essence, SUREL outperforms canonical GNNs by a great margin on prediction performance (almost 50% in all tasks).

2 PRELIMINARIES AND RELATED WORKS

In this section, we set up notations, formulate the SGRL problem and review some related works.

2.1 Notations

Definition 2.1 (Graph-structured data)**.** Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ denote an attributed graph, where $\mathcal{V} = [n]$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ are the node set and the edge set respectively. $X \in \mathbb{R}^{n \times d}$ denotes the node attributes with d-dimension. Further, we use $\mathcal{N}_v$ to represent the set of nodes in the direct neighborhood of node v, i.e., $\mathcal{N}_v = \{u : (u, v) \in \mathcal{E}\}$.

Definition 2.2 (m-hop Subgraph)**.** Given a graph $\mathcal{G}$ and a node set of interest Q, let $\mathcal{G}_Q^m$ denote the m-hop neighboring subgraph w.r.t the set Q. $\mathcal{G}_Q^m$ is the induced subgraph of $\mathcal{G}$, of which the node set $\mathcal{V}_Q^m$ includes the set Q and all the nodes in $\mathcal{G}$ whose shortest path distance to Q is less than or equal to m. Its edge set is a subset of $\mathcal{E}$, where each edge has both endpoints in its node set $\mathcal{V}_Q^m$. The nodes in $\mathcal{V}_Q^m$ still carry the original node attributes if $\mathcal{G}$ is attributed.

2.2 Graph Learning Problems and Background

Now, we formally formulate the GRL and SGRL problems.

Definition 2.3 (Graph Representation Learning (**GRL**))**.** Given a graph $\mathcal{G}$ and a queried set of nodes Q, graph representation learning aims to learn a mapping from the graph-structured data to some predicting labels as $f(\mathcal{G}, Q) \to y$, where the mapping $f(\mathcal{G}, Q)$ may reflect structures and node attributes of $\mathcal{G}$ and their relation to Q.

Definition 2.4 (Subgraph-based GRL (**SGRL**))**.** Given a node set Q over an ambient graph $\mathcal{G}$ and a positive integer m, SGRL is to learn the mapping to some labels, which takes the m-hop neighboring subgraph of Q in $\mathcal{G}$ as the input $f(\mathcal{G}_Q^m, Q) \to y$. An SGRL task typically is given some labeled node set queries $\{(Q_i, y_i)\}_{i=1}^{L}$ for training and other unlabeled node set queries $\{Q_i\}_{i=L+1}^{L+N}$ for testing.

We list a few important examples of SGRL tasks. **Link prediction** seeks to estimate the likelihood of a link between two endpoints in a given graph. Additionally, it can be generalized to predict the type of links, such as relation prediction for heterogeneous graphs. In this case, the set Q corresponds to a pair of nodes. The network scientific community has identified the importance of leveraging the local induced subgraphs for link prediction [21]. For example, the number of common friends (shown as neighbors in a social network) implies how likely two individuals may become

friends in the future. Another generalized form of link prediction is **higher-order pattern prediction**, where the set Q consists of three or more nodes. The goal is to predict whether the set of nodes in Q will foster a covered edge (termed hyperedge).

Graph neural networks (GNNs). Canonical GNNs associate each node v with a vector representation $\mathbf{h}$, which is learned and updated by aggregating messages from v's neighbors, as

$$\mathbf{h}_v^k = \text{UPDATE}\left(\mathbf{h}_v^{k-1}, \text{AGGREGATE}\left(\{\mathbf{h}_u^{k-1} | u \in \mathcal{N}_v\}\right)\right).$$

Here, UPDATE is implemented by neural networks while AGGREGATE is a pooling operation invariant to the order of the neighbors. By unfolding the neighborhood around each node, the computation graph to get each node representation forms a tree structure. According to Def. 2.4, canonical GNNs seem also able to perform SGRL by encoding the local subtree rooted at each node into a node representation (See Fig. 1a). Nevertheless, by this way, each node representation only *separately* reflects the subgraph around each node but cannot *jointly* represent the subgraph around multiple nodes, which yields the problem in Fig. 1. However, the SGRL framework considered in this work is able to learn the representation of the joint subgraph around a queried node set.

2.3 Other Related Works

Without exception, previous works focus on improving the scalability of canonical GNNs and their system support, but some of their techniques inspire the design of SUREL.

To overcome the memory bottleneck of GPU when processing large-scaled graphs, sub-sampling the graph structure is a widely adopted strategy. GraphSAGE [12] and VR-GCN [4] use uniform sampling schema and variance reduction technique respectively to restrict the size of node neighbors; PIN-SAGE [42] exploits Personalized PageRank (PPR) scores to sample neighbors. FastGCN [3] and ASGCN [15] perform independent layer-wise node sampling to allow neighborhood sharing. Cluster-GCN [6] and GraphSAINT [45] study subgraph-based mini-batching approaches to reduce the size of training graphs. Note that the subgraphs in our setting are substantially different from theirs, since our subgraphs work as features for queries while their subgraphs are a compensatory choice to achieve better scalability.

Many works better the system support for GNNs. DGL [37] and PyG [8] are designed for scalable single-machine GNN training. Marius [26] is proposed to efficiently learn large-scale graph embeddings on a single machine. There are several distributed systems dedicated to GNNs: AliGraph [41] addresses the storage issue of applying GNNs on massive industrial graphs; AGL [46] employs a subgraph-based system for GRL; ROC [16] builds a multi-GPU framework for deeper and larger GNN models; Dorylus [33] designs a CPU-based distributed system for GNN training. G^3 [22] speedups GNN training via supporting parallel graph-structured operations. Zhou et al. [51] uses feature dimension pruning to accelerate large-scale GNN inference. However, all these systems only support canonical GNNs so they all suffer from the intrinsic modeling limitations of GNNs.

3 THE ARCHITECTURE OF SUREL

In this section, we first give an overview of the SUREL framework as shown in Fig. 2. Then, we focus on the design and the implementation of three modules: Walk Sampler & Relative Position Encoder

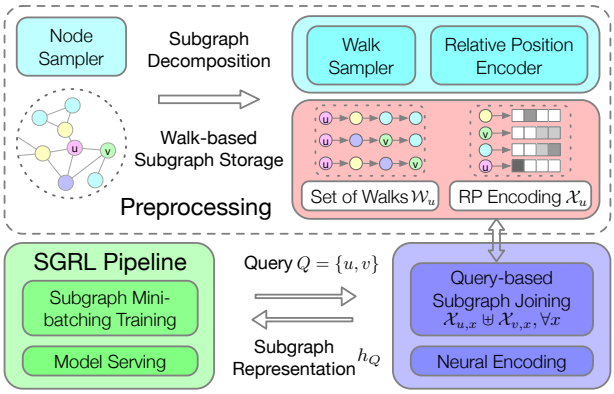


Figure 2: The System Architecture of Subgraph-based Graph Representation Learning Framework (SUREL).

(Preprocessing), Walk-based Subgraph Storage, Query-based Subgraph Joining & Neural Encoding. At last, we elaborate an efficient training pipeline with Subgraph Query Mini-batching.

3.1 Overview

Existing SGRL frameworks that extract a subgraph per query do not support efficient training and inference. m-hop subgraph extraction faces the size "explosion" issue as many nodes have significantly large degrees in real-world networks. Moreover, subgraphs of different sizes cause workload fluctuation, hindering load balancing and memory management.

Subgraph extraction can be replaced with efficient walk-based sampling, which sidesteps all above issues via regulating the number and the length of sampled walks. The number and the length of these walks are small constants, so the space and time complexity here is only linear w.r.t the number of nodes. Specifically, during preprocessing, SUREL reduces the subgraph around each node in a given graph to a set of random walks originated from it. To compensate for the loss of structural information after breaking subgraphs into walks, an intra-node distance feature termed relative positional encoding (RPE) is proposed, which enables locating each node in the sampled subgraph. The collected set of walks paired with its RPEs is hosted in the walk-based subgraph storage, with a dedicated data structure designed to support rapid and intensive access. The preprocessing flow is presented in the upper part of Fig. 2.

For training and testing, given a query (set of nodes), SUREL employs *subgraph joining* to implicitly construct a subgraph around the entire query in full parallel. First, all the walks originated from the queried node set are grouped. Then, the precomputed node-level RPEs are joined into query-level RPEs. SUREL further adopts neural networks to encode the grouped walks paired with query-level RPEs, and makes final predictions based on the obtained subgraph representation. A mini-batching strategy is designed to maximize data reuse during training by exploiting the query overlaps.

3.2 Preprocessing - Walk Sampling & Encoding

The bottleneck of current SGRL frameworks is how to cheaply acquire the m-hop neighbors for each queried set of nodes. SUREL proposes to decompose the m-hop subgraph into a set of m-length walks that start from the queried set of nodes. As the walks are regular, their storage and access are extremely efficient. This also

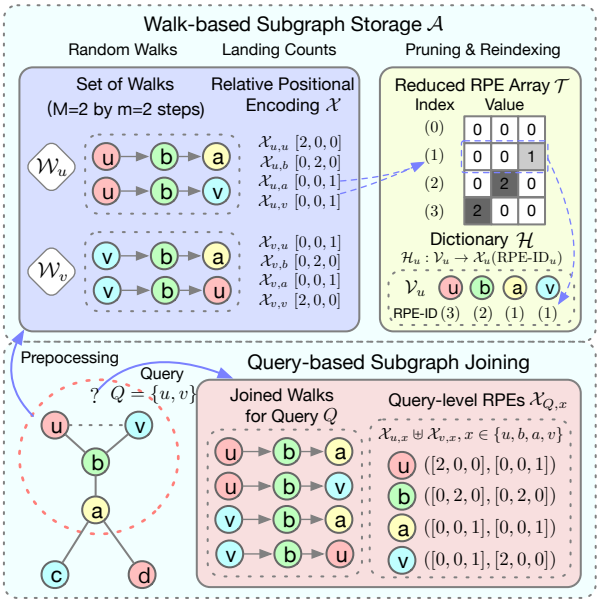


Figure 3: An Illustration of Joining RPE into Query-level RPEs with the Support of Walk-based Subgraph Storage.

resolves the computational problem caused by the long-tailed distribution of node degrees. More importantly, the collected walks grouped by their starting nodes can be shared and reused among different queries. Our design decouples SGRL from redundant subgraph extraction and enables the reusability of preprocessed data. We summarize the preprocessing routines with the support of hash-indexed storage in Algorithm 1 and introduce the specifics next.

Walk Sampling. During preprocessing, SUREL samples M-many m-step walks for every node in a given graph. As Fig. 3 (upper left) shows, the sampled walks are grouped in a set $\mathcal{W}_u$, where u denotes the starting node of these walks. Walk sampling can be easily divided into parallelizable pieces. The parallelization is implemented based on NumPy and OpenMP framework in C. Moreover, to further accelerate walk sampling, we use compressed sparse row (CSR) to represent the graph. The CSR format consists of two arrays, `idxptr` of length $|\mathcal{V}|+1$ used to record the degrees of nodes, and `indices` of size $|\mathcal{E}|$, each row of which corresponds to the neighbor list per node. CSR allows intensive fast access to the neighbors of a node while keeping the memory cost low, which is vital for walk sampling in large-scale graphs.

Relative Positional Encoding (RPE). Structural information gets lost after breaking subgraphs into walks. SUREL compensates such loss via RPE to locate the relative position of a node in each sampled subgraph, which characterizes the structural contribution of the node to its corresponding subgraph.

For each set of walks $\mathcal{W}_u$, we first establish a set $\mathcal{V}_u$ that contains distinct nodes appearing in $\mathcal{W}_u$. Define *node-level* RPE $\mathcal{X}_u : \mathcal{V}_u \to \mathbb{R}^{m+1}$ as follows: for each node $x \in \mathcal{V}_u$, a vector $\mathcal{X}_{u,x} \in \mathbb{R}^{m+1}$ is assigned, where $\mathcal{X}_{u,x}[i]$ is the landing counts of node x at position i in all walks of $\mathcal{W}_u$. In SUREL, RPE can be computed on the fly as walks get sampled, thus resulting in nearly zero extra computational cost. The set of walks $\mathcal{W}_u$ paired with the RPE $\mathcal{X}_u$ essentially characterize a sub-sampled subgraph around the node u. Next, we present a dedicated data structure to host $\mathcal{W}_u$ and $\mathcal{X}_u$ altogether.

Algorithm 1: Data Preprocessing in SUREL

Input: Graph $\mathcal{G}$; number of walks M; step of walks m
Output: Associative array $\mathcal{A}$, RPE array $\mathcal{T}$

Initialize the array $\mathcal{A}$ and $\mathcal{T}$, the dictionary $\mathcal{H}$
for *each node* $u \in \mathcal{G}$ **do**
  Run M times m-step random walks on $\mathcal{G}$ as a set of walk $\mathcal{W}_u \in \mathbb{Z}^{M\times m}$;
  Add the key $\mathcal{V}_u = \text{set}(\mathcal{W}_u)$ to $\mathcal{H}_u$;
  Calculate RPE for $\forall x \in \mathcal{V}_u$, save the value $\mathcal{X}_{u,x}$ to $\mathcal{T}$, and write its index in $\mathcal{T}$ as RPE-ID$_{u,x}$ back to $\mathcal{H}_u(x)$;
  Insert $\{u : (\mathcal{W}_u, \mathcal{H}_u)\}$ to $\mathcal{A}$
end
Prune $\mathcal{T}$ and update the value of $\mathcal{H}$ by re-indexing.

3.3 Walk-based Subgraph Storage

It is easy to manage the collected set of walks due to its regularity. An $m*M$-sized chunk is allocated to each set of walks, which assists to speed up data fetching. How to organize node-level RPE presents a real challenge because the cardinality of the set $|\mathcal{V}_u|$ varies from node to node. One naïve way to avoid such irregularity is to directly scatter these RPEs back to nodes in previously collected walks. But, this gives an $m * M * (m+1)$ tensor, resulting in an unrealizable memory need. Moreover, it loses track of node IDs in walks that are needed for joining subgraphs later.

We use an associative array $\mathcal{A}$ to organize all walk-based subgraphs as shown in the upper part of Fig. 3. For each node $u \in \mathcal{V}$, its corresponding entry in $\mathcal{A}$ is a node-level subgraph formed as a tuple $(\mathcal{W}_u, \mathcal{H}_u)$. Here, $\mathcal{W}_u$ is a set of walks starting from u, and $\mathcal{H}_u$ is a dictionary that maps the unique node set $\mathcal{V}_u$ of $\mathcal{W}_u$ to its corresponding node-level RPE $\mathcal{X}_u$. The use of dictionary resolves irregularities in $\mathcal{V}_u$ mentioned above, while maintaining the connection between node IDs and their RPEs. In addition, array $\mathcal{T}$ is introduced to store RPE values centrally, rather than scattered across dictionaries. As Fig. 3 (upper right) shows, the value of $\mathcal{H}_u(x)$ is now replaced with the index of the RPE value $\mathcal{X}_{u,x}$ stored in $\mathcal{T}$ accordingly, noted as RPE-ID$_{u,x}$. This design overall guarantees the access of RPE in $O(1)$ time.

The above $\mathcal{A}$ and $\mathcal{H}_u$ are built on top of `uthash`'s macros [1], with extended support for arbitrary insertions and deletions of key–value pairs. It offers data access and search in $O(1)$ time on average, which is about as good as the direct address table but greatly reduces the space wastage. In particular, it has no dependency or need for communication between multiple hash queries, thus can be pleasingly executed in parallel. Both $\mathcal{A}$ and $\mathcal{H}_u$ are stored in RAM on the CPU side. As we observed in Fig. 3, there are many repeated RPE values. Once all nodes are sampled, the array $\mathcal{T}$ can be pruned to remove duplicates. RPE-IDs will be updated synchronously when $\mathcal{T}$ is reindexed. For example, both node a and v have the RPE value of $[0,0,1]$, whose index in $\mathcal{T}$ is (1) after pruning. Thus, both $\mathcal{H}_u(a)$ and $\mathcal{H}_u(v)$ are assigned to the new RPE-ID as (1). The shape of $\mathcal{T}$ is regular and its size is usually small after pruning, which can be fully loaded in GPU. In practice, we found that pining RPEs in GPU memory is critical, as it can significantly reduce the communication cost of moving data back and forth between RAM and SDRAM.

[1] https://troydhanson.github.io/uthash/

3.4 Query-based Subgraph Joining

The storage designed above records the downsampled subgraph around each node. As SGRL is mostly useful for making predictions over a set of nodes Q, here we further illustrate how to get the joined subgraph around all the nodes $u \in Q$.

The idea is to concatenate all set of walks $[..., \mathcal{W}_u, ...]$ for $u \in Q$, since each set of walks $\mathcal{W}_u$ can be viewed as a subgraph around u. Besides, each node x in the walks will be paired with a *query-level* RPE $\mathcal{X}_{Q,x}$ that characterizes the relative position of node x in the joint subgraph around the queried set Q. Specifically, $\mathcal{X}_{Q,x}$ is defined by joining all RPEs $\mathcal{X}_{u,x}$ for $u \in Q$, i.e., $\mathcal{X}_{Q,x} = \uplus_{u \in Q} \mathcal{X}_{u,x} (\triangleq [..., \mathcal{X}_{u,x}, ...]) \in \mathbb{R}^{(m+1)\times|Q|}$. There will be some $u \in Q$ such that $x \notin \mathcal{V}_u$, for which $\mathcal{X}_{u,x}$ is set to all zeros. Through this procedure, the joined subgraph with query-level RPEs is sent to GPU for representation learning and then model inference.

The data structure described in Sec. 3.3 enables a highly parallel implementation of subgraph joining along with optimized memory management. On the CPU side, $\mathcal{X}_{Q,x}$ is not directly used to assemble walks. Instead, we use a query-level RPE-ID that joins node-level RPE indices in $\mathcal{T}$, i.e. use RPE-ID$_{Q,x} = [..., \text{RPE-ID}_{u,x}, ...] \in \mathbb{R}^{|Q|}$ for $u \in Q$, which reduces the memory cost from $(m+1) * |Q|$ to $|Q|$. For instance, in Fig. 3 (bottom right), $\mathcal{X}_{Q,u} = ([2, 0, 0], [0, 0, 1])$ can be substituted by RPE-ID$_{Q,u} = (3, 1)$, as their RPE values locate at the entry (3) and (1) of $\mathcal{T}$. As follows, SUREL pre-allocates an array with the fixed-size $[m * M * |Q|, |Q|]$, where $m * M * |Q|$ is the size of walks around Q. Then, SUREL fills the index array with RPE-ID$_{u,x}$ by multithreads. Note that RPE-ID$_{u,x}$ can be rapidly retrieved via the dictionary operation $\mathcal{H}_u(x)$. Lastly, assembling RPE values to walks is performed on GPU via the indexing operation $\mathcal{X}_{u,x} = \mathcal{T}(\text{RPE-ID}_{u,x})$, where $\mathcal{T}$ is pinned in GPU memory earlier. SUREL incorporates a Python/C hybrid API for subgraph joining, building on top of NumPy, PyTorch, OpenMP and `uthash`.

Some remarks can be made here. First, the above algorithm contains some redundancy to compute the query-level RPE-ID for the nodes that appear multiple times in the walks. In practice, we find that about half of the nodes appear only once, thus doubling the computation time at most. To avoid such redundancy, one can first compute the set union $\mathcal{V}_Q = \cup_{u \in Q} \mathcal{V}_u$, and then compute the query-level RPE-ID by traversing all nodes in $\mathcal{V}_Q$. However, parallel set union is difficult to implement efficiently. When multithreading is enabled, we observe a significant increase in the efficiency of SUREL, as opposed to the union operation. Also, by dynamically adjusting the number of threads, the workload between CPU and GPU can be well balanced. Second, we have empirically found that using RPE-ID instead of RPE to assemble walks provides an observable performance boost (speed up by 2× or more), otherwise data communication between CPU and GPU would the main bottleneck.

3.5 Neural Encoding

After subgraph joining for each query, the obtained subgraph is represented by a concatenated set of walks on which nodes are paired with query-level RPEs (See Fig. 3). Next, we introduce neural networks to encode these walks into a subgraph representation h_Q.

Due to its regularity, any sequential models, e.g., MLP, CNN, RNN, and transformers can be adopted for sampled walks. We test RNN and MLP for neural encoding, both of which achieve similar results. Next, we take the RNN as an example. We encode each walk $W = (w_0, w_1, ..., w_m) \in \mathcal{W}$ as $\text{enc}(W) = \text{RNN}(\{f(\mathcal{X}_{Q,w_i})\}_{i=0,1,...,m})$, where w_i's denote the node at step i in one sampled walk. Here, f is to encode the query-level RPE. Node or edge attributes for each step $w_k \in W$ can be supported by attaching those attributes after its RPE. To obtain the final subgraph representation of Q, we aggregate the encoding of all the associated walks through a mean pooling, i.e., $h_Q = \text{mean}(\{\text{enc}(W) | W \text{ starts from some } u \in Q\})$. In the end, a two-layer classifier is used to make prediction by taking h_Q as input. In our experiments, all the tasks can be formulated as binary classification, and thus we adopt Binary Cross Entropy as the loss function.

Algorithm 2: The Training Pipeline of SUREL

Input: A graph $\mathcal{G}$, a set of training queries $\{(Q_i, y_i)\}$, batch capacity B_1, batch size B_2
Output: A Neural Network for Neural Encoding NN$(\cdot)$

```
Prepare the collection of set of walks W and RPEs X
for iter = 1, ..., max_iter do
    Initialize the set Q = ∅ to track reached queries;
    Randomly choose a seed-set of nodes V̄ from ∪Q;
    Run breath-first search to expand V̄ and Q until
      |V̄| = B_1 or |Q| = B_2;
    Generate negative training queries (if not given) for a
      mini-batch and put them into Q;
    Perform subgraph joining for queries in Q;
    Encode the concatenated walks by NN(·) to get the
      subgraph representation h_Q for each query;
    Use backpropagation [29] to optimize model parameters.
end
```

3.6 The Training and Serving Pipelines

SUREL organically incorporates the storage designed in Sec. 3.3 and the subgraph-joining operation described in Sec. 3.4 to achieve efficient training and model serving.

Subgraph queries Q's are sets of nodes, which often come from a common ambient on a large graph. There might be many overlaps between different queries and their m-hop induced subgraphs. If the queried subgraphs are known in prior, we may put these queries with high node overlap into the same batch to improve data reuse. Here, queries of each given task are assumed to have the same size, e.g. $|Q| = 2$ for link prediction. In practice, test queries are usually given online while the training ones can be prepared in advance. Hence, we propose to accelerate the training pipeline by mini-batching the overlapping queries. Practitioners can choose the appropriate pipeline according to the specific situation. Algorithm 2 summarizes the overall training procedure of SUREL.

Mini-batching for Training. We first randomly sample a seed-set of nodes $\bar{\mathcal{V}}$ from the union of queried node sets $\cup Q$. Then, we run breadth-first search (BFS) to expand the seed-set $\bar{\mathcal{V}}$. Neighbor fetching of the BFS here is based on the grouped queries instead of the original graph: a neighbor of node u is defined as the node that shares at least one query with it. During BFS, the reached queries will be added to a set $\mathcal{Q}$. The expansion stops once the size of either the seed-set $\bar{\mathcal{V}}$ or the mini-batch $\mathcal{Q}$ reaches some

Table 1: Summary Statistics for Evaluation Datasets.

Dataset	Type	#Nodes	#Edges
citation2	Homo.	2,927,963	30,561,187
collab	Homo.	235,868	1,285,465
ppa	Homo.	576,289	30,326,273
ogb-mag	Hetero.	Paper(P): 736,389 Author(A): 1,134,649	P-A: 7,145,660 P-P: 5,416,271
tags-math	Higher.	1,629	91,685 (projected) 822,059 (hyperedges)
DBLP-coauthor	Higher.	1,924,991	7,904,336 (projected) 3,700,067 (hyperedges)

pre-defined limits. Since the data structure for each query in SUREL after subgraph joining is regular, it is easy to decide the size limits of seed-set and mini-batch based on resource availability (i.e. GPU memory). In practice, this BFS procedure improves reusability of data within each mini-batch, and may significantly decrease the communication cost between CPU and GPU. If the training set only contains positive queries (often in link/motif prediction tasks), we design an efficient sampling strategy for negative queries by the same principle that randomly pairs them within the same batch.

4 EVALUATION

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

- Regarding prediction performance, can SUREL outperform state-of-the-art SGRL models? Can SUREL significantly outperform canonical GNNs and transductive graph embedding methods due to the claimed benefit of SGRL?
- Regarding runtime, can SUREL significantly outperform state-of-the-art SGRL models? Can SUREL achieve runtime performance comparable to canonical GNNs? Previous SGRL models are typically much slower than canonical GNNs.
- How about the parameter sensitivity of SUREL? How do the parameters m and M impact the overall performance?
- How is the parallel design of SUREL performing and scaling?

4.1 Evaluation Setup

We conduct extensive experiments to evaluate the proposed framework with three kinds of graphs (homogeneous, heterogeneous, and higher-order homogeneous) on three corresponding types of tasks, namely, link prediction, relation prediction and higher-order pattern prediction. Homogeneous graphs are the graphs without node/link types. Heterogeneous graphs include node/link types. Higher-order graphs contain higher-order links that may connect more than 2 nodes. The dataset statistics are summarized in Table 1, most of which are larger than the datasets used in [50, 51], not to mention that our node-set prediction task is much more complex than the node classification task considered in the previous works.

Open Graph Benchmark (OGB). We use three link prediction and one relation prediction datasets [13]: ppa - a protein interaction network, collab - a collaboration network, and citation2 - a citation network; and one heterogeneous network ogb-mag, which contains four types of nodes (paper, author, institution and field) and their relations extracted from MAG [36].

Higher-order Graph Dataset. DBLP-coauthor is a temporal higher-order network that records co-authorship of papers as timestamped higher-order links. tags-math contains sets of tags that are applied to questions on the website math.stackexchange.com as higher-order links. For the two higher-order graphs, SUREL and all the baselines will treat them as standard graphs by projecting higher-order links into cliques. However, the training and test queries are generated based on higher-order links detailed next.

Table 2: Comparison of SGRL Methods for Subgraph Sampling. Suppose using $O(|\mathcal{E}|)$ many queries and S to denote the average size of sampled subgraphs. The wall-clock time is measured on citation2 test set with $p = 16$ threads.

Methods	SEAL (1-hop) [47, 49]	DE-GNN [20]	SUREL
Time Complexity	$O(S\|\mathcal{E}\|)$	$O(S\|\mathcal{E}\|)$	$O(\frac{mM}{p} \cdot \|\mathcal{V}\|)$
Wall Time	36h	> 1 month	26s

Settings. For *Link Prediction*, we follow the data split as OGB requires to isolating the validation and test links (queries) from the graphs. For *Relation Prediction*, the relations of paper-author (P-A) and paper-citation (P-P) are selected. The dataset is split based on timestamps. 0.5% of existing edges of each target relation type are selected from ogb-mag. For each paper, two authors/citations are picked from its P-A/P-P relations respectively, one for validation and the other for testing. The remaining links are used for training. For *Higher-order Pattern Prediction*, we focus on predicting whether two nodes will be connected to a third node concurrently via a higher-order link in the future. Specifically, positive queries are node triplets, where two nodes are linked before the timestamp t and the third node establishes connection to the pair via a higher-order link after t. The split ratio of positive node triplets is 60/20/20 for training/validation/testing. For *Relation Prediction* and *Higher-order Pattern Prediction*, each positive query is paired with 1000 randomly sampled negative queries (except tags-math uses 100) in testing. For fair comparison, all baselines are tested with the same set of negative queries sampled individually for each dataset. All experiments are run 10 times independently, and we report the mean performance and standard deviations.

Baselines. We consider three classes of baselines. *Graph Embedding methods* for transductive learning: Node2vec [10] and DeepWalk [28], which learns a single embedding for each node and may suffer from the information over-squashing issue; *Canonical GNNs*: GCN [18], GraphSAGE [12], GraphSAINT [45], Cluster-GCN [6], Relational GCN (R-GCN) [30], Relation-aware Heterogeneous Graph Neural Network (R-HGNN) [43]; *SGRL models*: SEAL[47, 49], DE-GNN[20]. SEAL supports both offline and online subgraph extraction per query. However, it takes SEAL 2+ hours and 102GB RAM to offline extract 2% training subgraphs on citation2. Thus, we only keep the online setting for SEAL. DE-GNN only supports offline subgraph extraction. Table 2 compares subgraph sampling for different SGRL methods. We adopt official implementations of above baselines with tuned parameters that match reported results. SUREL uses an 2-layer MLP for embeddings of RPEs and an 2-layer RNN to encode query-level joined walks. The obtained subgraph embeddings are fed into an MLP classifier for final prediction. Default training parameters are: learning rate lr=1e-3 with early stopping of 5-epoch patience, dropout p=0.1, Adam [17] as the optimizer, batch capacity $B_1 = 1500$, and batch size $B_2 = 32$. Hidden dimension d and walk parameters M, m are investigated in Sec. 4.4. Detailed parameter configurations can be found in the attached artifact and Appendix D of the arxiv version of this work [link].

Table 3: Results for Link Prediction, Relation Prediction, and Higher-order Pattern Prediction.

Models	citation2 MRR (%)	collab Hits@50 (%)	ppa Hits@100 (%)
Node2vec	61.28±0.15	47.54±0.78	18.05±0.52
DeepWalk	84.47±0.04	49.08±0.93	27.80±1.71
GCN	84.74±0.21	44.75±1.07	18.67±1.32
SAGE	82.60±0.36	54.63±1.12	16.55±2.40
Cluster-GCN	80.04±0.25	44.02±1.37	3.56±0.40
GraphSAINT	79.85±0.40	53.12±0.52	3.83±1.33
SEAL	87.67±0.32	**63.64±0.71**	48.80±3.16
SUREL	**89.74±0.18**	63.34±0.52	**53.23±1.03**

Models	MAG(P-A) MRR (%)	MAG(P-P) MRR (%)	tags-math MRR (%)	DBLP-coauthor MRR (%)
GCN	39.43±0.29	57.43±0.30	51.64±0.27	37.95±2.59
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	-	-
R-HGNN	33.41±2.47	45.91±3.28	-	-
DE-GNN	-	-	36.67±1.59	Timeout
SUREL	**45.33±2.94**	**82.47±0.26**	**71.86±2.15**	**97.66±2.89**

Metric. The evaluation metrics include Hits@K and Mean Reciprocal Rank (MRR). Hit@K counts the percentage of positive samples ranked at the top-K place against all the negative ones. MRR firstly calculates the inverse of the ranking of the first correct prediction against the given number of paired negative samples, and then an average is taken over the total queries.

Environment. We use a server with four Intel Xeon Gold 6248R CPUs, 1TB DRAM, and eight NVIDIA RTX 6000 (24GB) GPUs.

4.2 Prediction Performance Analysis

Table 3 shows results of three prediction tasks. Apparently, for these three link prediction benchmarks, the performance of SGRL models is significantly better than transductive graph embedding models and canonical GNNs, particularly for the challenging tasks over ppa and collab. Within SGRL models, SUREL sets two SOTA results on ppa and citation2, and gets comparable performance on collab against SEAL, which validates the modeling effectiveness of our proposed walk-based framework. For relation prediction and higher-order pattern prediction, we observe a large gap (up to 60%) between canonical GNNs and SUREL-based models, especially in higher-order cases. This again demonstrates the inherent modeling limitation of canonical GNNs to predict over a set of nodes. DE-GNN suffers from serious scalability issues when employing subgraph extraction for higher-order pattern prediction. Our best attempt is to deploy DE-GNN on tags-math by using 10% training samples, while the other three graphs failed. DE-GNN spends more than 300 hours preprocessing just 5% training queries of DBLP-coauthor.

4.3 Runtime and Memory Complexity Analysis

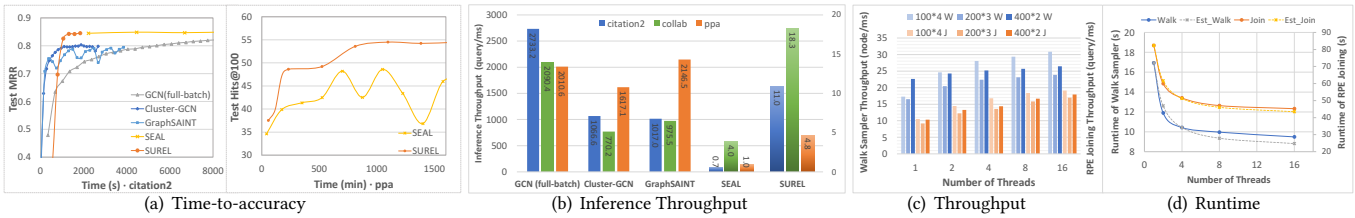
Table 4 reports the runtime, memory consumption comparison on a single machine (using one GPU) between canonical GNNs and SGRL models. SUREL offers a reasonable total runtime on these benchmarks compared with canonical GNNs. Meanwhile, its preprocessing overhead is negligible as showed in Table 4 under the term '**Prep.**', and the higher-order case can be efficiently handled as well. SEAL adopts online extraction, and thus the cost is not counted in preprocessing, while its training suffers from the computation bottleneck. DE-GNN uses offline extraction, and it takes 15+ hours and 98GB RAM to process training queries in tags-math, which is obviously incapable of scaling to DBLP-coauthor (so not present in Table 4). Overall, SUREL substantially accelerates the subgraph extraction and makes it feasible for SGRL on large-scale graphs.

Table 4: Breakdown of Runtime, Memory Consumption for Different Models on citation2, collab and DBLP-coauthor. Training time is calculated if no better validation result is observed in 3 consecutive epochs, which assumes the model has converged. Full-batch training models need NVIDIA A100 (48GB) GPUs, results of which are marked with *. Other models take less time on A100 than on RTX 6000.

	Models	Runtime (s)				Memory (GB)	
		Prep.	Train	Inf.	Total	RAM	SDRAM
citation2	**GCN** *	17	16,835	32	16,884	9.5	37.55
	Cluster-GCN	197	2,663	82	2,942	18.3	14.07
	GraphSAINT	140	3,845	86	4,071	16.9	14.77
	SEAL (1-hop)	46	22,296	130,312	152,654	36.5	3.35
	SUREL	31	2,096	7,959	10,086	15.2	4.50
collab	**GCN**	6	840	0.1	846	3.2	5.17
	Cluster-GCN	8	649	0.2	666	3.4	5.29
	GraphSAINT	<1	6,746	0.2	6,747	3.2	6.58
	SEAL (1-hop)	10	7,675	37	7,722	15.4	6.97
	SUREL	<1	1,720	8	1,728	3.6	5.57
DBLP	**GCN** *	-	153	95	248	8.0	25.80
	SAGE *	-	86	77	161	7.5	24.70
	SUREL	10	430	1,667	2,107	8.6	8.61

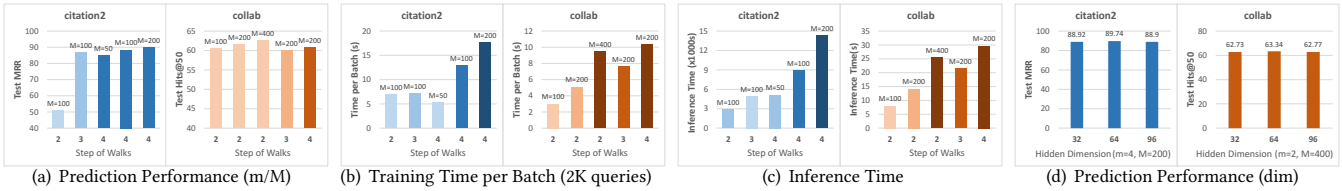
In terms of memory management, SUREL achieves comparable RAM usage to canonical GNNs, because the number of walks M and the steps m are small constants in practice. The extra memory cost is linear in $|\mathcal{V}|$, so the total memory cost is still dominated by the original graph. However, SEAL induces much more RAM usage as it extracts subgraphs of long-tail sizes, and its total memory cost is often super-linear in $|\mathcal{V}|$. Both SEAL and SUREL consume much less SDRAM because they do not need GPU to load large adjacency matrices and host node representations.

We further profile the training and inference performance, and present it in the left of Fig. 4. The time-to-accuracy comparison between canonical GNNs and SGRL models is shown in Fig. 4(a). Each dot indicates one training epoch for full-batch GCN, SEAL and SUREL, 10 training epochs for Cluster-GCN and GraphSAINT. As it shows, both SEAL and SUREL use 1-3 epochs to get good enough performance, and each epoch of SUREL takes around 1/10 time of SEAL on citation2. The time per epoch of full-batch GCN is comparable with SUREL, while Cluster-GCN and GraphSAINT are faster. However, these models generally take longer time to converge to even subpar performance. On ppa, the curve of SEAL is pretty oscillating, leading to longer convergence. SUREL uses large M and m to achieve better and more stable performance on ppa, so the training time per epoch is comparable with SEAL. The training curves of canonical GNN baselines are not plotted for ppa because of their poor performance (See Table 3).



(a) Time-to-accuracy (b) Inference Throughput (c) Throughput (d) Runtime

Figure 4: (a-b) Performance Profiling of Training & Inference; (c-d) Performance Scaling of SUREL (*W*alk Sampler and Query-level RPE *J*oining) against Different Number of Threads.



(a) Prediction Performance (m/M) (b) Training Time per Batch (2K queries) (c) Inference Time (d) Prediction Performance (dim)

Figure 5: Hyper-parameter Analysis: the number of walks M, the step of walks m, and the hidden dimension d.

Fig. 4(b) provides the comparison of end-to-end inference throughput between two classes of models. Canonical GNNs offer rapid inference, since they generate node representations as the intermediate computation results that are shared across all queries. But as aforementioned, sharing node representations may over-squash useful information and degenerate performance as shown in Table 3. SEAL, as SGRL, achieves good prediction performance but its inference is extremely slow, because of subgraph extraction per query. SUREL fundamentally solves this bottleneck by replacing the extraction with walk-based subgraph joining. It is $\mathbf{4-16}\times$ faster than SEAL on inference for link prediction, and achieves even more speedup than DE-GNN in higher-order settings.

4.4 Significant Hyperparameter Analysis

The number M, the step m of walks and the hidden dimension d effect scalability and accuracy of SUREL. To examine their impact, we evaluate SUREL on `citation2`, a large sparse graph, and `collab`, a medium dense graph, for different values of m, M, and d.

Prediction Performance. Fig. 5(a) and 5(d) show the prediction results. As expected, the performance consistently increases if we use a larger number of walks M. But for the step m, it is not always true that longer steps will guarantee better results, which depends on the specifics of the dataset. For instance, in network `citation2`, to accurately predict the link between two papers, more steps are needed as it would capture a larger group of papers which share similar semantics. While for `collab`, the case is different, as deeper walks would include more noise for predicting collaborations between two authors. In general, some small m ($2 \sim 5$) and M ($50 \sim 400$) ensure adequate performance. By adjusting m and M, we can achieve the trade-off between accuracy and scalability, none of which is achievable through other SGRL models. Moreover, SUREL is insensitive to the hidden dimension as shown by Fig. 5(d).

Training and Inference Time Cost. As Figs. 5(b) and 5(c) demonstrated, the time of walk sampling and subgraph joining is nearly linear w.r.t. the total number of walks ($m * M$) under the same number of threads (16 by default). Here, we do not regulate M based on the degree of each node in a query, which may induce certain duplication in sampled walks originated from the nodes with small degrees. Using degree-adaptive M is promising to further improve the scalability of SUREL while keeping good prediction performance. We leave such investigation for future study.

4.5 Performance Scaling

To investigate the scaling performance of the parallel implementation, we examine the runtime of heavy operations in SUREL by using different numbers of threads. Fig. 4(c) shows the throughput of walk sampler and query-level RPE joining on `citation2`. The runtime is also compared to the estimated runtime by Amdahl's law [9] shown in Fig. 4(d): walk sampling and RPE joining are in good agreement with the expected speedup, thus implying well parallelized implementation.

5 CONCLUSION

We propose a novel computational paradigm, SUREL for subgraph-based representation learning on large-scale graphs. SUREL targets predicting relations over set of nodes. It decouples graph structures into sets of walks to avoid irregularities in subgraphs and enable reuse of intermediate results. It then applies walk-based subgraph joining paired with relative positional encoding for representation learning of queried node sets. Such design allows for full parallelization and significantly improves model scalability. SUREL incorporates the principle of algorithm and system co-design that unlocks the full potential of learning on large-scale data with limited resources. To the best of our knowledge, this is the first work to study subgraph-based representation learning from the perspective of system scalability. Experiments also show that SUREL achieves superior performance in both prediction and scalability on three different SGRL tasks over six large, real-world graph benchmarks.

ACKNOWLEDGMENTS

We greatly thank all the reviewers for valuable feedback and actionable suggestions. H. Yin and P. Li are supported by the 2021 JPMorgan Faculty Award and the National Science Foundation (NSF) award HDR-2117997.

REFERENCES

[1] Uri Alon and Eran Yahav. 2020. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*.

[2] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. 2020. Improving graph neural network expressivity via subgraph isomorphism counting. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*.

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

[4] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*. PMLR, 942–950.

[5] Zhengdao Chen, Lei Chen, Villar Soledad, and Joan Bruna. 2020. Can Graph Neural Networks Count Substructures?. In *Advances in Neural Information Processing Systems*, Vol. 33.

[6] 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 & Data Mining*. 257–266.

[7] Alessandro Epasto and Bryan Perozzi. 2019. Is a single embedding enough? Learning node representations that capture multiple social contexts. In *Proceedings of the Web Conference 2019*. Association for Computing Machinery, 394–404.

[8] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*.

[9] Ananth Grama, Vipin Kumar, Anshul Gupta, and George Karypis. 2003. *Introduction to parallel computing*. Pearson Education.

[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 855–864.

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

[12] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.

[13] 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. *arXiv preprint arXiv:2005.00687* (2020).

[14] Kexin Huang and Marinka Zitnik. 2020. Graph meta learning via local subgraphs. In *Advances in Neural Information Processing Systems*, Vol. 33.

[15] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. In *Advances in Neural Information Processing Systems*, Vol. 31.

[16] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Proceedings of Machine Learning and Systems* 2 (2020), 187–198.

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

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

[19] 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.

[20] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning. In *Advances in Neural Information Processing Systems*, Vol. 33.

[21] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58, 7 (2007), 1019–1031.

[22] Husong Liu, Shengliang Lu, Xinyu Chen, and Bingsheng He. 2020. G3: when graph neural networks meet parallel graph processing systems on GPUs. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2813–2816.

[23] 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 & Data Mining*. 1959–1969.

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

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

[26] Jason Mohoney, Roger Waleffe, Henry Xu, Theodoros Rekatsinas, and Shivaram Venkataraman. 2021. Marius: Learning Massive Graph Embeddings on a Single Machine. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. 533–549.

[27] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.

[28] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 701–710.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[30] 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.

[31] Balasubramaniam Srinivasan and Bruno Ribeiro. 2020. On the Equivalence between Node Embeddings and Structural Graph Representations. In *International Conference on Learning Representations*.

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

[33] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. 495–514.

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

[35] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. 2022. Equivariant and Stable Positional Encoding for More Powerful Graph Neural Networks. In *International Conference on Learning Representations*.

[36] 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.

[37] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*.

[38] Xiyuan Wang and Muhan Zhang. 2022. GLASS: GNN with Labeling Tricks for Subgraph Representation Learning. In *International Conference on Learning Representations*.

[39] 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*.

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

[41] Hongxia Yang. 2019. Aligraph: A comprehensive graph neural network platform. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3165–3166.

[42] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[43] Le Yu, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. 2022. Heterogeneous graph representation learning with relation awareness. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[44] 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. In *Advances in Neural Information Processing Systems*, Vol. 34.

[45] 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*.

[46] Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. 2020. AGL: a scalable system for industrial-purpose graph machine learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3125–3137.

[47] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, Vol. 31.

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

[49] 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. In *Advances in Neural Information Processing Systems*, Vol. 34.

[50] Wentao Zhang, Zhi Yang, Yexin Wang, Yu Shen, Yang Li, Liang Wang, and Bin Cui. 2021. GRAIN: Improving Data Efficiency of GraPh Neural Networks via Diversified inFluence Maximization. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2473–2482.

[51] Hongkuan Zhou, Ajitesh Srivastava, Hanqing Zeng, Rajgopal Kannan, and Viktor Prasanna. 2021. Accelerating Large Scale Real-Time GNN Inference Using Channel Pruning. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1597–1605.