SketchNE: Embedding Billion-Scale Networks Accurately in One Hour

Yuyang Xie, Yuxiao Dong, *Senior Member, IEEE*, Jiezhong Qiu, Wenjian Yu, *Senior Member, IEEE*, Xu Feng, and Jie Tang, *Fellow, IEEE*

***Abstract*—We study large-scale network embedding with the goal of generating high-quality embeddings for networks with more than 1 billion vertices and 100 billion edges. Recent attempts LightNE and NetSMF propose to sparsify and factorize the (dense) NetMF matrix for embedding large networks, where NetMF is a theoretically-grounded network embedding method. However, there is a trade-off between their embeddings' quality and scalability due to their expensive memory requirements, making embeddings less effective under real-world memory constraints. Therefore, we present the SketchNE model, a scalable, effective, and memory-efficient network embedding solution developed for a single machine with CPU only. The main idea of SketchNE is to avoid the explicit construction and factorization of the NetMF matrix either sparsely or densely when producing the embeddings through the proposed sparse-sign randomized single-pass SVD algorithm. We conduct extensive experiments on nine datasets of various sizes for vertex classification and link prediction, demonstrating the consistent outperformance of SketchNE over state-of-the-art baselines in terms of both effectiveness and efficiency. SketchNE costs only *1.0 hours* to embed the Hyperlink2012 network with *3.5 billion* vertices and *225 billion* edges on a CPU-only single machine with embedding superiority (e.g., a *282%* relative HITS@10 gain over LightNE).**

***Index Terms*—Memory-efficient, network embedding, network representation learning, randomized matrix factorization.**

I. Introduction

Representation learning on graphs has recently provided a new paradigm for modeling real-world networks [1]. Learning structural representations for networks, i.e., network embedding, aims to map network entities into a latent space. The learned entity embeddings have been used to power various billion-scale online services, such as DeepWalk [2] in Alibaba [3], LINE [4] in LinkedIn [5], metapath2vec [6] and NetSMF [7] in Microsoft Academic [8], PinSage in Pinterest [9].

Manuscript received 26 May 2022; revised 7 December 2022; accepted 23 February 2023. Date of publication 1 March 2023; date of current version 15 September 2023. This work was supported by NSFC under Grant 61872206, in part by the National Key R&D Program of China under Grant 2018YFB1402600, in part by NSFC for Distinguished Young Scholar under Grant 61825602, in part by NSFC under Grants 61836013 and 62276148, and in part by Zhipu.AI. The work of Jiezhong Qiu was supported in part by Zhejiang Lab under Grant 2022PE0AC04. Recommended for acceptance by Z. Guan. *(Corresponding author: Wenjian Yu.)*

The authors are with the Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing 100084, China (e-mail: xyy18@mails.tsinghua.edu.cn; ericdongyx@gmail.com; xptree@foxmail.com; yu-wj@tsinghua.edu.cn; fx17@mails.tsinghua.edu.cn; jietang@tsinghua.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TKDE.2023.3250703, provided by the authors.

Digital Object Identifier 10.1109/TKDE.2023.3250703

Take Facebook for example, it leverages the word2vec [10] based graph embedding system [11] to learn structural embeddings for its 3 billion user base. These embeddings are then consumed in various downstream applications. To maintain the quality of these embeddings, it is required to periodically embed such networks as its underlying structure consistently evolves, ideally as frequently as possible, e.g., every few hours in Alibaba [3]. However, according to our estimates, state-of-the-art (SOTA) graph embedding systems, i.e., GraphVite [12]—a DeepWalk [2] based system—and PyTorch-BigGraph [11]—would cost days if not weeks by using powerful CPU and GPU clusters to embed a network of 3B users.

Though skip-gram based embedding models, e.g., DeepWalk [2], LINE [4], and metapath2vec [6], have been widely adopted in large-scale solutions. They are still limited to handle billion-scale networks at speed, as discussed above. Recently, a theoretical study demonstrates that these models can be transformed as implicit factorization of a closed-form matrix [13]. Based on this discovery, the NetMF model is proposed to explicitly construct and factorize the matrix that is implicitly factorized by DeepWalk, i.e., the NetMF matrix[1] $f^{\circ}(\boldsymbol{M})$, in which $\boldsymbol{M}$ can be approximated by $\boldsymbol{L}$ and $\boldsymbol{R}$—the two matrices formed by the eigen-decomposition over the graph Laplacian of a given network—and $f^{\circ}(\cdot)$ is an element-wise logarithm function. Additionally, addressing the matrix form $f^{\circ}(\boldsymbol{LR})$ can benefit various machine learning scenarios, such as the attention mechanism in Transformer [14]—softmax$(\cdot)$, the linear layer with ReLU [15] activation—ReLU$(\cdot)$, and the kernel method [16].

Despite its outperformance over skip-gram based methods, it is computationally prohibitive for NetMF to handle million-scale networks as it needs to construct and factorize $f^{\circ}(\boldsymbol{M})$, which is an $n \times n$ dense matrix with n being the number of vertices. To address this, one recent attempt NetSMF [7] proposes to construct a sparse version of $f^{\circ}(\boldsymbol{M})$ by a graph spectral based sampling technique and then leverage sparse matrix factorization to produce vertex embeddings. More recently, LightNE [17] advances NetSMF by further reducing its sampling cost, utilizing other system-wise optimizations, and borrowing the spectral propagation strategy from ProNE [18]. In doing so, LightNE outperforms SOTA systems, including NetSMF, ProNE, GraphVite, and PyTorch-BigGraph, in terms of both computational cost and embedding effectiveness.

[1]The detailed NetMF matrix $f^{\circ}(\boldsymbol{M})$ can be found in Table I

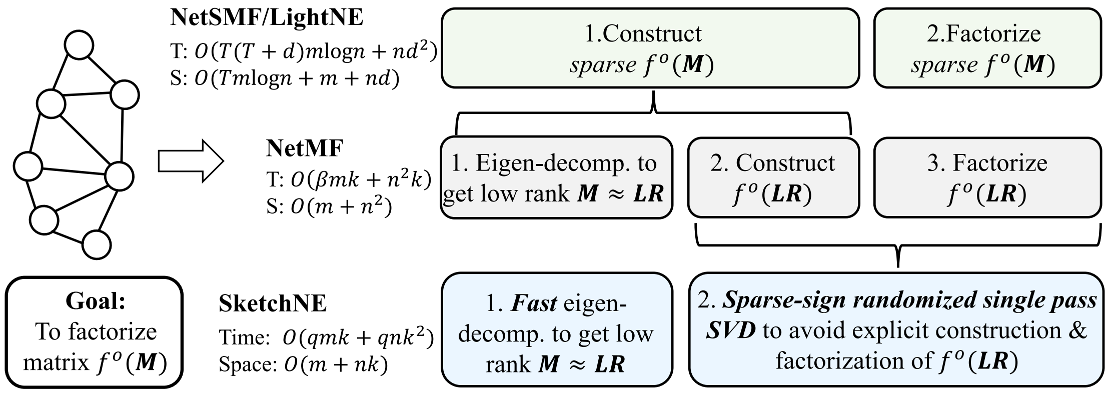

NetSMF/LightNE
T: $O(T(T+d)m\log n + nd^2)$
S: $O(Tm\log n + m + nd)$
1.Construct sparse $f^o(\boldsymbol{M})$
2.Factorize sparse $f^o(\boldsymbol{M})$
NetMF
T: $O(\beta mk + n^2k)$
S: $O(m+n^2)$
1. Eigen-decomp. to get low rank $\boldsymbol{M} \approx \boldsymbol{LR}$
2. Construct $f^o(\boldsymbol{LR})$
3. Factorize $f^o(\boldsymbol{LR})$
Goal: To factorize matrix $f^o(\boldsymbol{M})$
SketchNE
Time: $O(qmk + qnk^2)$
Space: $O(m+nk)$
1. Fast eigen-decomp. to get low rank $\boldsymbol{M} \approx \boldsymbol{LR}$
2. Sparse-sign randomized single pass SVD to avoid explicit construction & factorization of $f^o(\boldsymbol{LR})$

Fig. 1. The overview of SketchNE versus NetMF and NetSMF/LightNE. The symbols used are listed in Table I.

However, the performance of LightNE and NetSMF heavily relies on the number of samplings that directly corresponds to the memory cost, that is, more samplings make the sparse matrix more close to $f^\circ(\boldsymbol{M})$ and thus yield better embeddings, while consuming more memory. For example, to generate competitive embeddings for the OAG data [17] of 67 M vertices and 895 M edges, LightNE requires 1493 GB memory space to have a sufficient number of samples. In order to embed larger networks, such as those of billions of vertices, LightNE has to sacrifice the quality of the embeddings under the real-world memory constraint.

Contributions. In light of the limitations of existing large-scale graph embedding solutions, the goal of this work is to learn effective embeddings for billion-scale networks efficiently under certain memory constraints, e.g., to embed networks with 3B vertices and 200B edges within 1 h by using a single machine with 1500 GB memory. To achieve this, we present the SketchNE [2] model, an effective, scalable, and memory-efficient method for billion-scale network embedding. Fig. 1 illustrates the two technical components in SketchNE—a fast eigen-decomposition algorithm and a sparse-sign randomized single-pass SVD, each of which addresses the computational challenges in NetMF correspondingly.

First, we propose to factorize the target matrix $f^\circ(\boldsymbol{M})$ without explicitly constructing it, avoiding the direct or sparse construction and factorization in NetMF or NetSMF/LightNE. To achieve this, we present the sparse-sign randomized single-pass SVD algorithm by leveraging the concept of the randomized sketch matrix.

Second, the step above still requires $\boldsymbol{L}$ and $\boldsymbol{R}$, though the explicit construction of $f^\circ(\boldsymbol{M})$ is not demanded anymore. Thus, we further introduce a fast randomized eigen-decomposition algorithm to approximate the computation of $\boldsymbol{L}$ and $\boldsymbol{R}$ and give an upper bound of the approximation error. Empirical tests show that we can achieve about 90× speedup over the original eigen-decomposition module in NetMF without performance loss on (small) networks that NetMF can actually handle.

Third, we combine the spectral propagation strategy which is proposed in [18] to further improve the quality of the inital embedding. We optimize our system for shared-memory architectures with Graph Based Benchmark Suite (GBBS) [19],

[2]The code is publicly available at https://github.com/xyyphant0m/SketchNE

TABLE I
SYMBOL USED THROUGHOUT THIS PAPER

Symbol	Description	Symbol	Description
G	input network	b	#negative samples
V	vertex set, $\lvert V\rvert = n$	T	context window size
E	edge set, $\lvert E\rvert = m$	$\boldsymbol{E}$	embedding matrix
$\boldsymbol{A}$	adjacency matrix	d	embedding dimension
vol(G)	volume of G	$\boldsymbol{U}_k$	truncated eigenvectors
$\boldsymbol{D}$	degree matrix	$\boldsymbol{\Lambda}_k$	truncated eigenvalues
$\boldsymbol{D}^{-\alpha}\boldsymbol{A}\boldsymbol{D}^{-\alpha}$	modified Laplacian	k	rank parameter
$\mathcal{L}$	normalized Laplacian	q	power parameter
$\boldsymbol{M}$	$\frac{\mathrm{vol}(G)}{bT}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r\boldsymbol{D}^{-1}$	$\boldsymbol{Y}$	sketch of $f^\circ(\boldsymbol{LR})$
s	oversampling parameter	z	column density

which has already shown its superiority when handling real-world networks with hundreds of billions of edges on a single machine. Intel Math Kernel Library (MKL) is used in SketchNE for basic linear algebra operations.

We conduct extensive experiments to examine the performance of SketchNE, including its effectiveness, efficiency, and memory cost. Specifically, we test SketchNE and other SOTA models/systems on five datasets for vertex classification and four datasets for link prediction. The results show that by using the least running time and memory among SOTA models/systems, SketchNE can consistently outperform nine large-scale baselines across five datasets for vertex classification and also offers significant improvements over LightNE on three billion-scale networks for link prediction. Notably, *SketchNE can embed the Hyperlink2012 network with 3.5 billion vertices and 225 billion edges in 1.0 hours by using 1,321 GB memory on a single machine, and the learned embeddings offer a 282% relative HITS@10 improvement over LightNE on the link prediction task.*

II. NetMF and Its Challenges

Given an undirected network $G = (V, E, \boldsymbol{A})$ with n vertices, m edges, adjacency matrix $\boldsymbol{A}$, degree matrix $\boldsymbol{D}$ and volume $\mathrm{vol}(G) = \sum_i\sum_j \boldsymbol{A}_{ij}$, the goal of network embedding is to learn an embedding matrix $\boldsymbol{E} \in \mathbb{R}^{n\times d}$ so that row i captures the structural property of vertex i [2], [4], [13], [20]. The embeddings can be then fed into downstream applications. The symbols are listed in Table I.

Many network embedding algorithms are based on random walk and skip-gram techniques, such as DeepWalk [2], LINE [4],

Algorithm 1: NetMF Under the new Formulation.

```
Input: adjacency matrix $\boldsymbol{A}$, rank $k$, embedding dimension $d$
Output: An embedding matrix $\boldsymbol{E} \in \mathbb{R}^{n\times d}$
1 $[\boldsymbol{U}_k, \boldsymbol{\Lambda}_k] = \text{eigs}(\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}, k)$ // Eigen-decomposition
2 $\boldsymbol{L} = \frac{\text{vol}(G)}{bT}\boldsymbol{D}^{-1/2}\boldsymbol{U}_k$, $\boldsymbol{R} = (\sum_{r=1}^{T}\boldsymbol{\Lambda}_k^r)\boldsymbol{U}_k^\top\boldsymbol{D}^{-1/2}$
3 $[\boldsymbol{U}_d, \boldsymbol{\Sigma}_d, \boldsymbol{V}_d] = \text{svds}(f^\circ(\boldsymbol{L}\boldsymbol{R}), d)$ // Rank-$d$ truncated SVD
4 return $\boldsymbol{E} = \boldsymbol{U}_d\boldsymbol{\Sigma}_d^{1/2}$ as network embedding
```

and node2vec [20]. Take DeepWalk for example, the vertex sequences traversed by random walkers are fed into the skip-gram model, which is usually parameterized by the context window size T and the number of negative samples b. Notably, these techniques are later shown to be theoretically equivalent to matrix factorization [13]. Based on this result, the NetMF algorithm is proposed to explicitly construct and factorize the matrix that is implicitly factorized by DeepWalk, namely the NetMF matrix:

$$\text{trunc\_log}^\circ\left(\frac{\text{vol}(G)}{bT}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r\boldsymbol{D}^{-1}\right), \tag{1}$$

where trunc_log° denotes the element-wise truncated logarithm, i.e., applying $\text{trunc\_log}(\text{x}) = \max(0, \log(\text{x}))$ to each entry of a matrix. However, the explicit construction and factorization of this matrix usually consumes $O(n^3)$ time as it tends to be a dense matrix even with a small T. To reduce time complexity, NetMF conducts truncated eigen-decomposition such that $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \approx \boldsymbol{U}_k\boldsymbol{\Lambda}_k\boldsymbol{U}_k^\top$, and factorizes the following approximate matrix of (1):

$$\text{trunc}_\text{log}^\circ\left(\frac{\text{vol(G)}}{\text{bT}}\boldsymbol{D}^{-1/2}\boldsymbol{U}_\text{k}\left(\sum_{\text{r=1}}^{\text{T}}\boldsymbol{\Lambda}_\text{k}^\text{r}\right)\boldsymbol{U}_\text{k}^\top\boldsymbol{D}^{-1/2}\right). \tag{2}$$

Reformulate the Goal of NetMF. With the above description, we can reformulate and generalize the goal of NetMF as follows:

Problem 1. Truncated SVD for element-wise function of a low-rank matrix.

Given: Two low-rank matrices $\boldsymbol{L} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{R} \in \mathbb{R}^{k\times n}$, a function $f : \mathbb{R} \to \mathbb{R}$ applied to each entry of $\boldsymbol{L}\boldsymbol{R}$, and desired dimensionality d.

Goal: Compute the rank-d truncated SVD for $f^\circ(\boldsymbol{L}\boldsymbol{R})$ such that:

$$[\boldsymbol{U}_d, \boldsymbol{\Sigma}_d, \boldsymbol{V}_d] = \text{svds}(f^\circ(\boldsymbol{L}\boldsymbol{R}), d). \tag{3}$$

In NetMF, $\boldsymbol{L} = \frac{\text{vol}(G)}{bT}\boldsymbol{D}^{-1/2}\boldsymbol{U}_k$, $\boldsymbol{R} = (\sum_{r=1}^{T}\boldsymbol{\Lambda}_k^r)\boldsymbol{U}_k^\top\boldsymbol{D}^{-1/2}$ and $f(\cdot) = \text{trunc\_log}(\cdot)$.

Algorithm 1 describes NetMF under the above new formulation. Unfortunately, it is still not capable of handling large networks, even the YouTube dataset with 1.1 million vertices used in DeepWalk [2], [21], mainly due to the following two challenges presented in Problem 1.

Challenge One: How to Solve svds$(f^\circ(\boldsymbol{L}\boldsymbol{R}), d)$ Efficiently? The major challenge lies in the requirement to explicitly construct and factorize $f^\circ(\boldsymbol{L}\boldsymbol{R})$, even after NetMF's attempt to perform the truncated eigen-decomposition. In fact, its construction and factorization in Algorithm 1 Line 3 demand $O(n^2)$ memory cost and $O(n^2\,k)$ time cost, making it computationally infeasible for large networks. It is worth noting that the element-wise truncated logarithm is very important to embedding quality and cannot be omitted [7], [13], [17], [22]. Otherwise, the embeddings can be realized without constructing the dense form, as in NRP [23], RandNE [24], and FastRP [25].

Challenge Two: How to Factorize $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ Efficiently? Although one may think the truncated eigen-decomposition of $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ (Line 1 of Algorithm 1) is a simple and efficient step, previous work [26], [27] and our analysis show that it is in fact computationally very expensive. In particular, Cohen-Steiner et al. [26] shows it is very difficult to obtain the spectrum of a large graph, and the eigen-decomposition of a large graph Laplacian [27] is practically very slow. The cost of the truncated eigen-decomposition [28] is $O(\beta mk)$, where $\beta \geq 1$ and its value depends on the convergence speed. The convergence, in turn, depends on the relative eigenvalue gap, making this constant term very big and this operation practically very expensive. This problem seems not prominent in NetMF because the datasets in its paper [13] are relatively small (the largest one is Flickr with 80 K vertices). However, for a slightly larger dataset YouTube with 1.1 M vertices, we observe that the SciPy implementation `eigsh` cannot finish the computation in three days, not to mention for billion-scale networks.

III. The SketchNE Model

In this section, we present SketchNE for embedding billion-scale networks at speed. In Section III-A, we propose a spare-sign single-pass SVD algorithm to resolve challenge one—factorizing $f^\circ(\boldsymbol{L}\boldsymbol{R})$ without constructing its dense form. In Section III-B, we introduce a fast randomized eigen-decomposition algorithm to resolve challenge two—accelerating the computation of $\boldsymbol{L}$ and $\boldsymbol{R}$. Section III-C combines the two sketching techniques and presents the overall algorithm.

A. Sketching-Based svds$(f^\circ(\boldsymbol{L}\boldsymbol{R}), d)$ Without Explicitly Computing and Factorizing $f^\circ(\boldsymbol{L}\boldsymbol{R})$

In this part, we formally introduce a sketch-based solution to svds$(f^\circ(\boldsymbol{L}\boldsymbol{R}), d)$ without explicitly computing and factorizing it.

Basic Randomized SVD. To solve the computational challenge of svds$(f^\circ(\boldsymbol{L}\boldsymbol{R}), d)$, we first revisit the randomized SVD in Algorithm 2, where $\boldsymbol{\Omega}$ (Line 2) is a Gaussian random matrix, s (Line 2) is the oversampling parameter for better accuracy, q (Line 5) is the power iteration index, and orth$(\cdot)$ (Lines 4, 6 and 7) is the orthonormalization operation which is usually implemented with QR factorization. The random projection in Line 3 generates a sketch matrix $\boldsymbol{Y}$, which identifies a subspace that captures dominant information of the input matrix

Algorithm 2: The Basic Randomized SVD.

```
Procedure basic_randomized_SVD ($\boldsymbol{X}, k, q$)
    $\boldsymbol{\Omega} = \mathrm{randn}(n,\ k+s)$    // oversampling parameter $s$: 10 or 20
    $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\Omega}$    // sketch matrix $\mathbb{R}^{n\times(k+s)}$
    $\boldsymbol{Q} = \mathrm{orth}(\boldsymbol{Y})$    // QR factorization
    for $i = 1, ..., q$ do    // power iteration (optional)
        $\boldsymbol{T} = \mathrm{orth}(\boldsymbol{X}^\top\boldsymbol{Q})$    // QR factorization
        $\boldsymbol{Q} = \mathrm{orth}(\boldsymbol{X}\boldsymbol{T})$    // QR factorization
    $\boldsymbol{B} = \boldsymbol{Q}^\top\boldsymbol{X}$    // reduced matrix $B \in \mathbb{R}^{(k+s)\times n}$
    $[\hat{\boldsymbol{U}}, \hat{\boldsymbol{\Sigma}}, \hat{\boldsymbol{V}}] = \mathrm{svd}(\boldsymbol{B})$    // Full SVD
    $\boldsymbol{U} = \boldsymbol{Q}\hat{\boldsymbol{U}}(:, 1:k), \boldsymbol{\Sigma} = \hat{\boldsymbol{\Sigma}}(1:k, 1:k), \boldsymbol{V} = \hat{\boldsymbol{V}}(:, 1:k)$
    return $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}$
```

$\boldsymbol{X} = f^\circ(\boldsymbol{LR})$. Then, with the subspace's orthonormal basis matrix $\boldsymbol{Q}$ computed in Line 4, one obtains the approximation $\boldsymbol{X} \approx \boldsymbol{Q}\boldsymbol{Q}^\top\boldsymbol{X}$ [29]. Lines 5∼8 are the power iteration scheme which is optional to improve the approximation accuracy. Next, Line 9 constructs a reduced matrix $\boldsymbol{B}$ by projecting the input matrix $\boldsymbol{X}$ to the subspace with orthonormal basis $\boldsymbol{Q}$. Finally, the approximate truncated SVD of $\boldsymbol{X}$ is obtained through performing SVD on matrix $\boldsymbol{B}$ in Lines 10-11.

We can see that there are four places in Algorithm 2 (Lines 3, 6, 7 and 9) requiring the explicit construction of $\boldsymbol{X} = f^\circ(\boldsymbol{LR})$. For Lines 6-7, we can avoid them by skipping the optional power iteration. We discuss how to deal with issues raised by Lines 3 and 9, respectively.

Issue One (Algorithm 2 Line 3). Sketching $f^\circ(\boldsymbol{LR})$. Computing sketch matrix $\boldsymbol{Y}$ requires matrix multiplication between $\boldsymbol{X} = f^\circ(\boldsymbol{LR})$ and a random matrix $\boldsymbol{\Omega}$. However, $f^\circ(\boldsymbol{LR})$ cannot be explicitly computed due to its $O(n^2 k)$ time complexity and $O(n^2)$ memory cost.

Solution: The Sparse-Sign Matrix for Sketching. We introduce the concept of the sparse-sign matrix to help solve the multiplication $\boldsymbol{X}\boldsymbol{\Omega} = f^\circ(\boldsymbol{LR})\boldsymbol{\Omega}$. The sparse-sign matrix, with similar performance to the Gaussian matrix, is another type of randomized dimension reduction map [30]. Algorithm 3 describes how to generate a sparse-sign matrix. To have a sparse-sign matrix $\boldsymbol{S} \in \mathbb{R}^{n\times h}$ with sketch size $h = d + s$, where d is the embedding dimension, and s is the oversampling parameter, we fix a column density parameter z in the range $2 \le z \le n$. We independently generate the columns of the matrix at random. For each column, we draw z i.i.d random signs and place them in z uniformly random coordinates as shown in Algorithm 3 Lines 5∼6. Line 8 is to get the unique row coordinates. According to [30], $z = \min(n, 8)$ will usually be a good choice.

After generating the sparse-sign matrix $\boldsymbol{S}$, we can use it as the random matrix, which multiplies $f^\circ(\boldsymbol{LR})$ on its right side to obtain the sketch matrix $\boldsymbol{Y}$. Considering that a column of the sparse-sign matrix $\boldsymbol{S}$ only has z nonzeros, it will generate at most $z \times h$ coordinates in the range of $[1, n]$ and $z \times h \ll n$. Therefore, we can perform column sampling according to these unique coordinates $\boldsymbol{p}$. Assuming that $v = \mathrm{size}(\boldsymbol{p}, 1)$, then v satisfy $v \le z \times h \ll n$. Based on the fact that $\boldsymbol{S}$ has only v non-zero

Algorithm 3: Generate a Sparse-Sign Matrix.

```
Procedure gen_sparse_sign_matrix ($n, h, z$)
    $\boldsymbol{S} = \mathrm{zeros}(n,\ h)$    // sketch size $h$
    $\boldsymbol{p} = \mathrm{zeros}(zh,\ 1)$    // column density parameter $z$
    for $j = 1, 2, ..., h$ do
        $\boldsymbol{p}((z(j-1)+1) : zj) = \mathrm{randperm}(n, z)$
        $\boldsymbol{S}(\boldsymbol{p}((z(j-1)+1) : zj), j) = \mathrm{sign}(\mathrm{randn}(z, 1))$
    $\boldsymbol{p} = \mathrm{unique}(\boldsymbol{p})$    // make coordinates unique
    return $\boldsymbol{S}, \boldsymbol{p}$
```

rows, we can immediately observe that we can have a sampling matrix $\boldsymbol{R}(:, \boldsymbol{p})$. Therefore, computing $\boldsymbol{Y} = f^\circ(\boldsymbol{LR})\boldsymbol{S}$ is exactly equivalent to

$$\boldsymbol{Y} = f^\circ\left(\boldsymbol{LR}(:, \boldsymbol{p})\right)\boldsymbol{S}. \tag{4}$$

The time cost of (4) is $O(nvk + nvh)$ and the memory cost is $O(nk + nv)$. However, when calculating $\boldsymbol{LR}(:, \boldsymbol{p})$ for a network with billions of vertices, it will introduce $O(nv)$ memory cost, which is still infeasible. To solve this, we adopt the batch matrix multiplication by selecting the fixed-size rows of $\mathbf{L}$ in turn to complete the multiplication, which further reduces the memory cost to $O(nk)$.

Issue Two (Algorithm 2 Line 9). Form the Reduced Matrix $\boldsymbol{B}$. According to Algorithm 2 Line 9, the reduced matrix $\boldsymbol{B}$ is constructed by $\boldsymbol{B} = \boldsymbol{Q}^\top f^\circ(\boldsymbol{LR})$, where $\boldsymbol{Q}$ is a dense matrix and $f^\circ(\boldsymbol{LR})$ is implicitly stored, making it too expensive to obtain $\boldsymbol{B}$.

Solution: The Randomized Single-Pass SVD. We leverage the idea of randomized single-pass SVD [30] to solve this issue. The basic idea is to obtain the approximate SVD results by visiting the target matrix $f^\circ(\boldsymbol{LR})$ only once. The process of single-pass SVD is as follows: First, we draw multiple sketch matrices that capture the row and column dominant information of matrix $f^\circ(\boldsymbol{LR})$ and compute SVD based on these sketch matrices. In [30], four sparse-sign random matrices $\boldsymbol{C} \in \mathbb{R}^{n\times h}, \boldsymbol{S} \in \mathbb{R}^{n\times h}, \boldsymbol{H} \in \mathbb{R}^{n\times l}, \boldsymbol{O} \in \mathbb{R}^{n\times l}$ are drawn for target matrix $f^\circ(\boldsymbol{LR})$. Then three sketch matrices

$$\boldsymbol{K} = f^\circ(\boldsymbol{LR})^\top\boldsymbol{C}, \boldsymbol{Y} = f^\circ(\boldsymbol{LR})\boldsymbol{S}, \boldsymbol{Z} = \boldsymbol{H}^\top f^\circ(\boldsymbol{LR})\boldsymbol{O} \tag{5}$$

are generated respectively. Second, we obtain the orthonormal matrices

$$\boldsymbol{P} = \mathrm{orth}(\boldsymbol{K}), \boldsymbol{Q} = \mathrm{orth}(\boldsymbol{Y}), \tag{6}$$

which capture the row and column dominant information of $f^\circ(\boldsymbol{LR})$, respectively. Then we get a great approximation as

$$f^\circ(\boldsymbol{LR}) \approx \boldsymbol{Q}\boldsymbol{Q}^\top f^\circ(\boldsymbol{LR})\boldsymbol{P}\boldsymbol{P}^\top. \tag{7}$$

By updating $\boldsymbol{Z}$ with (7), we can have

$$\boldsymbol{Z} = \boldsymbol{H}^\top f^\circ(\boldsymbol{LR})\boldsymbol{O} \approx (\boldsymbol{H}^\top\boldsymbol{Q})(\boldsymbol{Q}^\top f^\circ(\boldsymbol{LR})\boldsymbol{P})(\boldsymbol{P}^\top\boldsymbol{O}). \tag{8}$$

Third, we get the reduced matrix

$$\boldsymbol{W} = \left(\boldsymbol{H}^\top\boldsymbol{Q}\right)^\dagger \boldsymbol{Z} \left(\boldsymbol{P}^\top\boldsymbol{O}\right)^\dagger \approx \boldsymbol{Q}^\top f^\circ(\boldsymbol{LR})\boldsymbol{P} \tag{9}$$

Algorithm 4: Sparse-Sign Randomized Single-Pass SVD.

```
Procedure sparse_sign_rand_single_pass_SVD ($\boldsymbol{L}, \boldsymbol{R}, d$)
    $h = d + s_1, l = d + s_2$    // sketch size $h, l$
    $[\boldsymbol{S}, \boldsymbol{p}] = \text{gen\_sparse\_sign\_matrix}(n, h, z)$    // Alg. 3
    $\boldsymbol{Y} = f^\circ(\boldsymbol{LR}(:, \boldsymbol{p}))\boldsymbol{S}$    // sketch matrix $\boldsymbol{Y} \in \mathbb{R}^{n\times h}$
    $\boldsymbol{Q} = \text{orth}(\boldsymbol{Y})$
    $[\boldsymbol{O}, \boldsymbol{p}] = \text{gen\_sparse\_sign\_matrix}(n, l, z)$    // Alg. 3
    $\boldsymbol{Z} = \boldsymbol{O}^\top f^\circ(\boldsymbol{L}(\boldsymbol{p}, :)\boldsymbol{R}(:, \boldsymbol{p}))\boldsymbol{O}$    // sketch matrix $\boldsymbol{Z} \in \mathbb{R}^{l\times l}$
    $\boldsymbol{W} = (\boldsymbol{O}^\top\boldsymbol{Q})^\dagger\boldsymbol{Z}(\boldsymbol{Q}^\top\boldsymbol{O})^\dagger$    // reduced matrix $\boldsymbol{W} \in \mathbb{R}^{h\times h}$
    $[\hat{\boldsymbol{U}}, \hat{\boldsymbol{\Sigma}}, \hat{\boldsymbol{V}}] = \text{svd}(\boldsymbol{W})$    // Full SVD
    $\boldsymbol{U} = \boldsymbol{Q}\hat{\boldsymbol{U}}(:, 1:d), \boldsymbol{\Sigma} = \hat{\boldsymbol{\Sigma}}(1:d, 1:d), \boldsymbol{V} = \boldsymbol{Q}\hat{\boldsymbol{V}}(:, 1:d)$
    return $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}$
```

Algorithm 5: Fast Randomized Eigen-Decomposition.

```
Procedure freigs ($\boldsymbol{X}, k, q$)
    $[\sim, n] = \text{size}(\boldsymbol{X}, 2)$
    $\boldsymbol{\Omega} = \text{randn}(n, k + s)$
    $\boldsymbol{Y} = \boldsymbol{X\Omega}$    // sketch matrix $\boldsymbol{Y} \in \mathbb{R}^{n\times(k+s)}$
    $[\boldsymbol{Q}, \sim, \sim] = \text{eigSVD}(\boldsymbol{Y})$    // fast orthonormalization
    for $i = 1, 2, ..., q$ do    // power iteration
        $[\boldsymbol{Q}, \sim, \sim] = \text{eigSVD}(\boldsymbol{XXQ})$
    $\boldsymbol{S} = \boldsymbol{Q}^\top\boldsymbol{XQ}$    // reduced matrix $\boldsymbol{S} \in \mathbb{R}^{(k+s)\times(k+s)}$
    $[\hat{\boldsymbol{U}}, \boldsymbol{\Lambda}] = \text{eig}(\boldsymbol{S})$    // Full eigen-decomposition
    $\boldsymbol{U} = \boldsymbol{Q}\hat{\boldsymbol{U}}$
    return $\boldsymbol{U}(:, 1:k), \boldsymbol{\Lambda}(1:k, 1:k)$
```

by solving the least-squares problem. Finally, it will form a low rank approximation of the target matrix $f^\circ(\boldsymbol{LR})$ via

$$f^\circ(\boldsymbol{LR}) \approx \boldsymbol{QWP}^\top, \tag{10}$$

and the approximate truncated SVD of $f^\circ(\boldsymbol{LR})$ can be derived from performing SVD on $\boldsymbol{W}$. Therefore, $f^\circ(\boldsymbol{LR})$ is required only in the sketching process, and the reduced matrix $\boldsymbol{W}$ is constructed only by sketch matrices and random matrices.

In addition, we note that $f^\circ(\boldsymbol{LR})$ is symmetric in NetMF, thus the row dominant information is equal to column dominant information which means we can omit $\boldsymbol{K}$ and $\boldsymbol{H}$. This enables us to replace $\boldsymbol{P}$ with $\boldsymbol{Q}$ in (7)–(10) and replace $\boldsymbol{H}$ with $\boldsymbol{O}$ in (5), (8), and (9). In other words, we can further simplify and improve the above randomized single-pass SVD process: When the multiplication $\boldsymbol{Y} = f^\circ(\boldsymbol{LR})\boldsymbol{S}$ is performed, we can simultaneously draw another sketch matrix $\boldsymbol{Z} = \boldsymbol{O}^\top f^\circ(\boldsymbol{LR})\boldsymbol{O}$ with a sparse-sign random matrix $\boldsymbol{O} \in \mathbb{R}^{n\times l}$.

Overall: Sparse-Sign Randomized Single-Pass SVD. By combining the sparse-sign random matrix with single-pass SVD, we propose a sparse-sign randomized single-pass SVD algorithm to avoid the explicit construction and factorization of $f^\circ(\boldsymbol{LR})$ as Algorithm 4.

In Algorithm 4, Line 8 generates the reduced matrix $\boldsymbol{W}$, which involves solving the least-squares problem twice. The first is to solve $(\boldsymbol{O}^\top\boldsymbol{Q})\boldsymbol{T} = \boldsymbol{Z}$ for the temporary matrix $\boldsymbol{T}$ and the second is to solve $(\boldsymbol{O}^\top\boldsymbol{Q})\boldsymbol{W}^\top = \boldsymbol{T}^\top$ for the reduced matrix $\boldsymbol{W}$. The matrix $\boldsymbol{O}^\top\boldsymbol{Q} \in \mathbb{R}^{l\times h}$ is well-conditioned when $l \gg h$ which suggests choosing $s_2 \gg s_1$. Lines 3 and 6 generate two sparse-sign random matrices. Considering that s_1 and s_2 are small numbers, the time cost of Line 4 is $O(nv(k+d))$ and Line 7 costs $O(v^2(k+d) + vd^2)$. Line 5 costs $O(nd^2)$, Lines 8∼9 cost $O(d^3)$, and Line 10 costs $O(nd^2)$. Therefore, the time complexity of Algorithm 4 is $O(n(vk + vd + d^2))$, linear to the number of vertices n, which is much more efficient than the explicit construction and factorization in NetMF ($O(n^2 k)$).

Overall, we present the sparse-sign randomized single-pass SVD to address the computational challenges in NetMF, which is the first attempt to introduce single-pass low-rank matrix factorization into network embedding. It not only solves the challenges of NetMF, but also gives a solution to the general problem of factorizing $f^\circ(\boldsymbol{LR})$. Recently, Han et al. [31] proposes polynomial tensor sketch for this problem, which combines a polynomial approximation of f (e.g., Taylor and Chebyshev expansion) with tensor sketch for approximating monomials of entries of $\boldsymbol{LR}$. We will see an ablation study which applies polynomial tensor sketch to NetMF in Section IV-D.

B. Fast Eigen-Decomposition Via Sketching

By now, we bypass the major bottleneck of Algorithm 1 (Lines 3) without explicitly computing $f^\circ(\boldsymbol{LR})$. However, the solution in Algorithm 4 still requires the separate $\boldsymbol{L}$ and $\boldsymbol{R}$ as input, which are computed by the eigen-decomposition on $\boldsymbol{D}^{-1/2}\boldsymbol{AD}^{-1/2}$ in Algorithm 1 Line 1. Though the truncated eigen-decomposition costs only $O(\beta mk)$ FLOPs in theory ($\beta \geq 1$) [28], it is in practice almost infeasible to handle large networks due to the big constants in its complexity. In fact, the computation of this step for the YouTube dataset with 1.1 million vertices cannot complete within three days by using the commonly-used `eigsh` implementation, while the goal of this work is to embed billion-scale networks efficiently, e.g., in one hour.

To address this practical challenge, we introduce a fast randomized eigen-decomposition method to approximate $\boldsymbol{D}^{-1/2}\boldsymbol{AD}^{-1/2}$. According to [29], the symmetric approximation formula should be $\boldsymbol{X} \approx \boldsymbol{QQ}^\top\boldsymbol{XQQ}^\top$ and the truncated eigen-decomposition result of $\boldsymbol{X}$ can be derived by performing eigen-decomposition on the small matrix $\boldsymbol{Q}^\top\boldsymbol{XQ}$. By combining the techniques of the power iteration scheme and acceleration strategy [32], the fast randomized eigen-decomposition can be described as Algorithm 5.

Practically, a good decomposition of $\boldsymbol{D}^{-1/2}\boldsymbol{AD}^{-1/2}$ by `freigs` requires a large q, increasing the time cost (see the experiment in Section IV-D). To balance the trade-off between effectiveness and efficiency, we propose to perform Algorithm 5 on a modified Laplacian matrix $\boldsymbol{D}^{-\alpha}\boldsymbol{AD}^{-\alpha}$, where $\alpha \in (0, 0.5]$. Therefore, we have $\boldsymbol{U}_k\boldsymbol{\Lambda}_k\boldsymbol{U}_k^\top \approx \boldsymbol{D}^{-\alpha}\boldsymbol{AD}^{-\alpha}$. It means $\boldsymbol{D}^{-1/2}\boldsymbol{AD}^{-1/2}$ is computed as $\boldsymbol{D}^{-1/2+\alpha}\boldsymbol{U}_k\boldsymbol{\Lambda}_k\boldsymbol{U}_k^\top\boldsymbol{D}^{-1/2+\alpha}$ approximately. We give an upper bound of the approximation error by Lemma 4 and its proof both in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2023.3250703. In doing so, (1) can be approximated by

$$f^{\circ}(\boldsymbol{L}'\boldsymbol{R}') = f^{\circ}\left(\frac{\text{vol}(G)}{bT}\boldsymbol{D}^{-1+\alpha}\boldsymbol{U}_k\boldsymbol{\Lambda}_k\left(\sum_{r=1}^{T}\boldsymbol{K}^{r-1}\right)\boldsymbol{U}_k^{\top}\boldsymbol{D}^{-1+\alpha}\right), \tag{11}$$

where $\boldsymbol{L}' = \frac{\text{vol}(G)}{bT}\boldsymbol{D}^{-1+\alpha}\boldsymbol{U}_k$, $\boldsymbol{K} = \boldsymbol{U}_k^{\top}\boldsymbol{D}^{-1+2\alpha}\boldsymbol{U}_k\boldsymbol{\Lambda}_k$, $\boldsymbol{R}' = \boldsymbol{\Lambda}_k(\sum_{r=1}^{T}\boldsymbol{K}^{r-1})\boldsymbol{U}_k^{\top}\boldsymbol{D}^{-1+\alpha}$ and $f(\cdot) = \text{trunc\_log}(\cdot)$. $\boldsymbol{K}$ is a $k \times k$ matrix, making the computation of $\sum_{r=1}^{T}\boldsymbol{K}^{r-1}$ cheap. We further give an upper bound of the approximation error between the NetMF matrix and $f^{\circ}(\boldsymbol{L}'\boldsymbol{R}')$ by the following theorem. We can see the approximation is better with a larger k.

Theorem 1. Suppose f° denotes trunc_log°, i.e., the element-wise truncated logarithm, $f^{\circ}(\boldsymbol{M})$ is the matrix in (1), and $f^{\circ}(\boldsymbol{L}'\boldsymbol{R}')$ is defined by (11) which includes the quantities obtained with Algorithm 5. Then,

$$\|f^{\circ}(\boldsymbol{M}) - f^{\circ}(\boldsymbol{L}'\boldsymbol{R}')\|_F \leq \frac{(1+\varepsilon)d_{min}^{-1+2\alpha}B}{(c-1)bT}$$

with high probability. Here $|\lambda_j|$ is the jth largest absolute value of eigenvalue of $\boldsymbol{D}^{-\alpha}\boldsymbol{A}\boldsymbol{D}^{-\alpha}$, $B = \sqrt{\sum_{j=k+1}^{n}|\lambda_j|^2((c^T-1)(1+\frac{n}{c-1})-nT)}\text{vol}(G)$, $c = n(\frac{d_{max}}{d_{min}})^{1-2\alpha}$. d_{min} and d_{max} are the minimum and maximum vertex degrees, respectively.

Proof. See Appendix A, available in the online supplemental material. □

In Algorithm 5, the "eigSVD" is used as the orthonormalization operation. Compared with the QR factorization, eigSVD is much faster especially when $n \gg (k+s)$ [32]. Since the oversampling parameter s is smaller than k, Lines 4∼5 cost $O(mk+nk^2)$. According to [32], Lines 6∼8 cost $O(q(mk+nk^2))$, Line 9 costs $O(mk+nk^2)$, Line 10 costs $O(k^3)$, and Line 11 costs $O(nk^2)$. Overall, the time complexity of Algorithm 5 is $O(q(mk+nk^2))$. However, the actual FLOPs of Algorithm 5 is far fewer than that of `eigsh`. In practice, our empirical tests suggest that by setting α=0.4, Algorithm 5 with a small q shows on average ∼90X speedup to `eigsh` on the small datasets that can be handled by `eigsh` without noticeable impacts on the eigenvalues computed and by extension on the embeddings learned (See Fig. 3(a)).

C. The Overall Algorithm

In Section III-A, we develop a sparse-sign randomized single-pass SVD algorithm (Algorithm 4) to solve svds$(f^{\circ}(\boldsymbol{L}\boldsymbol{R}), d)$ without the explicit computation and factorization of the full matrix $f^{\circ}(\boldsymbol{L}\boldsymbol{R})$. In Section III-B, we propose a fast randomized eigen-decomposition method to get $\boldsymbol{L}$ and $\boldsymbol{R}$ for large networks. Empowered by these two techniques, we address the two computational challenges faced by NetMF, respectively. To this end, we present the SketchNE algorithm to learn embeddings for billion-scale networks in Algorithm 6.

In Algorithm 6, Line 1 computes the fast randomized eigen-decomposition for the modified Laplacian matrix $\boldsymbol{D}^{-\alpha}\boldsymbol{A}\boldsymbol{D}^{-\alpha}$. Lines 2∼3 form the approximations of matrices $\boldsymbol{L}$ and $\boldsymbol{R}$. Line 4 is to compute the SVD of $f^{\circ}(\boldsymbol{L}'\boldsymbol{R}')$ by $\boldsymbol{L}'$ and $\boldsymbol{R}'$ through the sparse-sign randomized single-pass SVD. Then we form the inital embedding in Line 5. Line 6 conducts spectral propagation, which is a commonly-used and computationally-cheap enhancement technique [17], [18] that further improves embedding quality. c_r in Line 6 is the coefficients of Chebyshev polynomials, p is spectral propagation steps (the default setting is 10) and $\mathcal{L} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A}$ is normalized graph Laplacian matrix ($\boldsymbol{I}$ is the identity matrix).

Algorithm 6: SketchNE.

```
Input: A network $G = (V, E, \boldsymbol{A})$; Normalized parameter $\alpha$; rank parameter $k$; power parameter $q$; Embedding dimension $d$;
Output: An embedding matrix $\boldsymbol{E} \in \mathbb{R}^{n\times d}$
1 $[\boldsymbol{U}_k, \boldsymbol{\Lambda}_k] = \text{freigs}(\boldsymbol{D}^{-\alpha}\boldsymbol{A}\boldsymbol{D}^{-\alpha}, k, q)$ // Algorithm 5
2 $\boldsymbol{K} = \boldsymbol{U}_k^{\top}\boldsymbol{D}^{-1+2\alpha}\boldsymbol{U}_k\boldsymbol{\Lambda}_k$
3 $\boldsymbol{L}' = \frac{\text{vol}(G)}{bT}\boldsymbol{D}^{-1+\alpha}\boldsymbol{U}_k$, $\boldsymbol{R}' = \boldsymbol{\Lambda}_k(\sum_{r=1}^{T}\boldsymbol{K}^{r-1})\boldsymbol{U}_k^{\top}\boldsymbol{D}^{-1+\alpha}$
4 $[\boldsymbol{U}_d, \boldsymbol{\Sigma}_d, \sim] =$ sparse_sign_rand_single_pass_SVD$(\boldsymbol{L}', \boldsymbol{R}', d)$ // Algorithm 4
5 $\boldsymbol{E} = \boldsymbol{U}_d\boldsymbol{\Sigma}_d^{1/2}$ // inital embedding
6 $\boldsymbol{E} = \sum_{r=0}^{p} c_r\mathcal{L}^r\boldsymbol{E}$ // spectral propagation
7 return $\boldsymbol{E}$ as network embedding
```

Complexity of SketchNE. For Line 1, the input matrix $\boldsymbol{D}^{-\alpha}\boldsymbol{A}\boldsymbol{D}^{-\alpha}$ is still an $n \times n$ sparse matrix and has $2\,m$ nonzeros. According to Section III-B, it requires $O(q(mk+nk^2))$ time and $O(m+nk)$ space. As for Lines 2∼3, $O(nk^2)$ time and $O(nk)$ space are required. The time cost of Line 4 is $O(n(d^2+vk+vd))$ and its space cost is $O(n(k+d))$. Lines 5∼6 demand $O(pmd+nd)$ time and $O(m+nd)$ space. In total, the SketchNE has the time complexity of $O(q(mk+nk^2)+nd^2+nvk+nvd+pmd)$ and the space complexity of $O(m+nk)$. Therefore, there is a trade-off between efficiency and effectiveness on the choice of q. In practice, we can easily find q that offers clear superiority on both efficacy and efficiency, including both memory cost and computing time, over existing large-scale network embedding techniques. Consider that q, k, d, v, p are all very small compared to m and n, the overall time complexity of SketchNE is linear to the number of edges m and the number of vertices n.

D. Implementation Details

Memory Reduction. Considering the memory cost of SketchNE is $O(m+nk)$, while NetSMF/LightNE ties performance to memory cost. Therefore, we consider to further optimize the memory cost of SketchNE by the Graph Based Benchmark Suite (GBBS) [19], which is an extension of the Ligra [33] interface. We optimize the memory cost of SketchNE with the Graph Based Benchmark Suite (GBBS) [19], which is an extension of the Ligra [33] interface. The GBBS is easy to use and has already shown its practicality for many large scale fundamental graph problems. LightNE [17] introduces GBBS to network embedding problems and shows its superiority to real-world networks with hundreds of billions of edges. The main benefit of GBBS to SketchNE is the data compression. A sparse adjacency matrix is usually stored in the compressed sparse row (CSR) format, which is also regarded as an excellent compressed graph representation [34]. However, the CSR format still incurs a huge memory overhead for the networks with hundreds of billions of edges. For example, storing a network with 1 billion vertices and 100 billion edges costs 1121 GB memory. Therefore, we need to compress it further and reduce memory cost. The GBBS can be regarded as a compressed CSR format for the graph from Ligra+ [35], which supports fast parallel graph encoding and decoding.

Parallelization. Two major computational steps of the SketchNE are sparse matrix-matrix multiplication (SPMM) and matrix-matrix product (GEMM), which are well supported by the Intel MKL library. SPMM operation is well supported by MKL's Sparse BLAS Routine. However, MKL's Sparse BLAS Routine requires the sparse matrix in CSR format as the input, which contradicts the original intention of using GBBS. Fortunately, GBBS supports traversing all neighbors of a vertex u for the compressed CSR format, and we can propose an SPMM operation with the help of GBBS. In order to use GBBS to save memory cost, we propose a parallel GBBS-based SPMM operation to replace the SPMM operation in MKL's sparse BLAS routine. The parallel GBBS-based SPMM is implemented as follows. First, we traverse n vertices parallelly. Then, we traverse neighbor vertex v of vertex u to compute the quantity $\boldsymbol{D}(u,u)^{-\alpha}\boldsymbol{D}(v,v)^{-\alpha}$ corresponding to the sparse matrix. Finally, with the support of `cblas_saxpy` in MKL, we multiply the vth row of the row-major matrix with the quantity and add the result to the uth row of the target matrix. The SPMM operation based on GBBS is slightly slower than MKL's SPMM operation, but ensuring memory-efficient. The "eigSVD", "eig" and other operations in SketchNE are well supported by Intel MKL routines. Line 4 and Line 7 of the Algorithm 6 involve matrix column sampling and batch GEMM operation. They are easily parallelized with the "parallel for" derivative in OpenMP [36].

In conclusion, SketchNE is implemented in C++. We use GBBS to reduce memory usage and implement a GBBS-based SPMM operation. For better efficiency, we use the Intel MKL library for basic linear algebra operations and the OpenMP programming.

IV. Experiments

In this section, we evaluate SketchNE on multi-label vertex classification and link prediction tasks, following exactly the same experimental settings as existing studies [2], [4], [7], [11], [12], [13], [17], [20]. We introduce datasets in Section IV-A our experimental settings and results in Section IV-B and Section IV-C, respectively. The ablation and case studies is in Section IV-D.

A. Datasets

We employ five datasets for the multi-label vertex classification task. BlogCatalog and YouTube are small graphs with less than 10 million edges, while the others are large graphs with more than 10 million but less than 10 billion edges. For the link prediction task, We have four datasets in which vertex labels are not available. Livejournal is the large graph, while the others are very large graphs with more than 10B edges. These datasets are of different scales and but have been widely used in network embedding literature [7], [17]. The statistics of datasets are listed in Table II.

BlogCatalog [37] is a network of relationships of online users. The labels represent the interests of the users.

YouTube [21] is a video-sharing website, which allows user to upload, view, rate and share videos. The vertex labels represent the user's taste in the video.

Friendster-small [38] is a sub-graph induced by all the labeled vertices in Friendster. The vertex labels in this network are the same as those in Friendster.

Friendster [38] is a large social network in an online gaming site. For some of the vertices, they have labels representing the groups the user joined.

OAG [39] is a publicly available academic graph opened by Microsoft Academic [39] and AMiner.org [40]. The vertex labels represent the study fields of each author.

Livejournal [41] is an online blogging site, where users can follow others to form a large social network.

ClueWeb [42] was created to support research on information retrieval and related human language technologies. The links between webs form the very large graphs.

Hyperlink2014 [43] was extracted from the Common Crawl Corpus released in April 2014, covering 1.7 billion web pages and 124 billion hyperlinks between these pages.

Hyperlink2012 [43] was extracted from the 2012 Common Crawl Corpus covering 3.5 billion web pages and 225 billion hyperlinks between these pages.

B. Experimental Settings

Baselines and Hyper-Parameters Setting. We compare SketchNE with nine SOTA network embedding methods, including PyTorch-BigGraph (PBG) [11], GraphVite [12], NetMF [13], NetSMF [7], ProNE [18], LightNE [17], RandNE [24], FastRP [25] and NRP [23]. We also compare SketchNE with four GNN methods, including DGI [44], GraphCL [45], GCC [46] and GraphSAGE [47]. For all the baselines originally run on CPU and SketchNE, we test them with all the datasets with 88 threads on a server with two Intel Xeon E5-2699 v4 CPUs (88 virtual cores in total) and 1.5 TB memory. For GraphVite, we present the results obtained from the original paper (if existed), which uses a 4×P100 GPU server, and otherwise run it on a 4×V100 GPU server to get the results. For GNN methods, we run it on a server with a GeForce GTX 1080 Ti GPU to get the results. All the baselines are evaluated with the hyperparameters set default in the corresponding paper's GitHub Repository or tuned for the best performance. Their settings are as follows.

NetMF [13]. We download the authors' official source codes[3], and run experiments with default setting: $T = 10$, $k = 256$.

[3]https://github.com/xptree/NetMF

TABLE II
STATISTICS OF DATASETS

	Multi-label Vertex Classification Task					Link Prediction Task			
	BlogCatalog	YouTube	Friendster-small	Friendster	OAG	Livejournal	ClueWeb	Hyperlink2014	Hyperlink2012
$\|V\|$	10,312	1,138,499	7,944,949	65,608,376	67,768,244	4,847,571	978,408,098	1,724,573,718	3,563,602,789
$\|E\|$	333,983	2,990,443	447,219,610	1,806,067,142	895,368,962	68,993,773	74,744,358,622	124,141,874,032	225,840,663,232

RandNE [24]. We download the authors' official source codes[4], and follow the default hyper-parameter setting for BlogCatalog. For other datasets, we follow the suggestion of tuning hyper-parameters from the source codes. The order is from 1 to 3, and the weights are searched according to $w_{i+1} = \beta_i w_i$ where β_i is from $\{0.01, 0.1, 1, 10, 100\}$.

FastRP [25]. We download the authors' official source codes[5], and follow the authors' suggestion for hyper-parameter setting. α_1, α_2 and α_3 are set to $0, 0$ and 1, respectively. We use the official tuning script to tune α_4 and the normalization strength β. The search ranges for β and α_4 are $[-1, 0]$ and $[2^{-3}, 2^6]$, respectively.

NRP [23]. We download the authors' official source codes[6], and follow setting in [23]: $l_1 = 20, l_2 = 10, \alpha = 0.15, \epsilon = 0.2$, and $\lambda = 10$.

PBG [11]. We download the authors' official source codes[7], and run the example script for Livejournal. For other datasets, we run the codes with default 30 iterations and report the best result.

GraphVite [12]. We adopt the reported results for YouTube, Friendster-small and Friendster in the original paper [12]. For other datasets, we run the authors' official source codes[8] with default setting. For Livejournal, other methods select $d = 1024$, while the official implementation of GVT[9] only allows the selection of d up to 512.

NetSMF [7]. We download the authors' official source codes[10], and run with the default hyper-parameter setting.

ProNE [18]. We use the high-performance version of ProNE released by the LightNE GitHub Repository[11], and keep the hyper-parameters the same as those in [18]: $p = 10$, $\theta = 0.5, \mu = 0.2$.

LightNE [17]. We download the authors' official source codes[12], and run experiments with the default scripts and the parameter setting according to its original paper.

SketchNE. We set parameters $b = 1, z = 8, s_1 = 100, s_2 = 1000$ for all datasets and choose T equal 2, 5 or 10, except on the very large graphs where we set $s_1 = 0$. We follow the embedding dimension d setting in [11], [12], [17], and let the other baselines follow the same setting. The eigen-decomposition rank k should be larger than d. The parameters α and q affect the experimental results to a larger extent. The bigger q, the more accurate the eigen-decomposition and thus the better the performance. We tune α and q in the experiments, choosing α in the range $[0.35, 0.5]$ with step 0.05 and q in the range [5,30] with step 5. All parameter settings for SketchNE are listed in Table III.

TABLE III
HYPER-PARAMETERS FOR SKETCHNE

Datasets	T	k	q	α	s_1	s_2	d
BlogCatalog	10	256	20	0.5	100	1000	128
YouTube	10	256	30	0.35	100	1000	128
Friendster-small	2	256	2	0.35	100	1000	128
Friendster	2	128	2	0.4	100	1000	96
OAG	10	256	30	0.45	100	1000	128
Livejournal	5	1024	10	0.35	100	1000	1024
ClueWeb	5	32	6	0.45	0	1000	32
Hyperlink2014	5	32	10	0.45	0	1000	32
Hyperlink2012	5	16	5	0.45	0	1000	16

The Setting for GNN Methods. To compare SketchNE with GNN methods, we choose DGI [44], GraphCL [45], GCC [46] and GraphSAGE [47] as the baselines. We download the official source codes released in the original papers, and run with the default hyper-parameter setting. Considering the fact that most GNNs require vertex features as input, we follow the experimental setting in [48], which is to generate random features for vertices. Specifically, for DGI and GraphCL, we generate a 128-dimensional random vector following the Xavier uniform or normal distribution [49] for each vertex. However, for GraphSAGE, we use its "identity features" as suggested in its original paper (See Section III-B of [47]) where each vertex has a learnable representation. DGI, GraphCL and GraphSAGE are trained on each dataset in an unsupervised way, we obtain the embedding matrix after the training process of these methods converges. GCC is a pretrained GNN model which does not require training-from-scratch on downstream tasks, thus we download the pretrained model and then encode every vertex on our datasets to a 64-dimensional embedding.

The Setting for Vertex Classification. To facilitate a fair comparison, we follow the training ratio setting in [2], [7], [12], [17]. A portion of labeled vertices are sampled for training and the remaining are used for testing. We complete the task by using one-vs-rest logistic regression implemented by LIBLINEAR [50]. The prediction procedure is repeated five times and the average performance is evaluated in terms of both Micro-F1 and Macro-F1 [51].

[4]https://github.com/ZW-ZHANG/RandNE
[5]https://github.com/GTmac/FastRP
[6]https://github.com/AnryYang/NRP-code
[7]https://github.com/facebookresearch/PyTorch-BigGraph
[8]https://github.com/DeepGraphLearning/graphvite
[9]https://graphvite.io/docs/0.2.1/api/solver.html#graphvite.solver.GraphSolver
[10]https://github.com/xptree/NetSMF
[11]https://github.com/xptree/LightNE
[12]https://github.com/xptree/LightNE

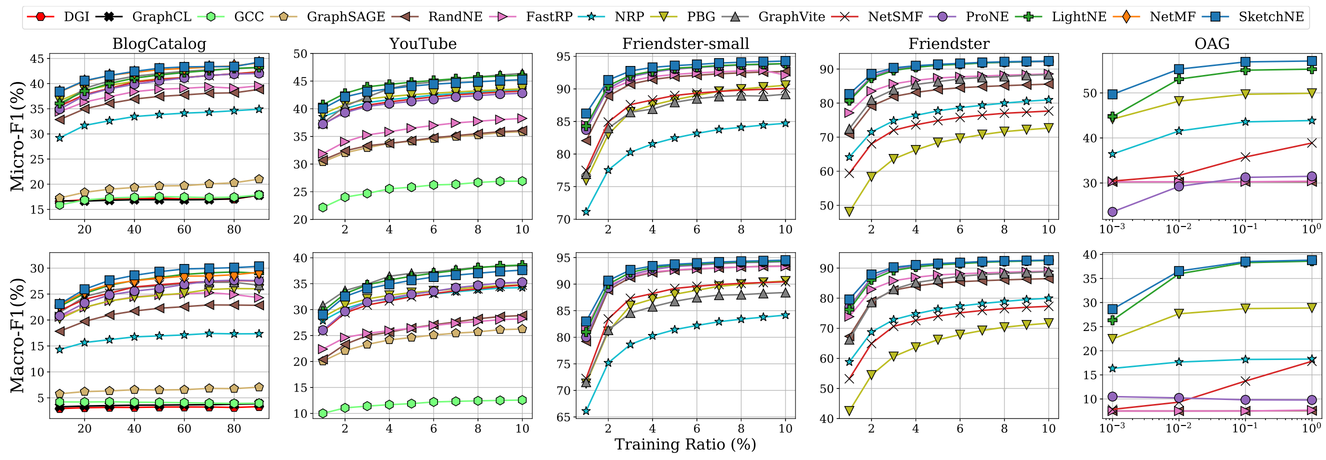


Fig. 2. Vertex classification performance (Micro-F1 and Macro-F1) w.r.t. the ratio of training data. For methods that cannot handle computation or cannot finish job in one day, the results are not available and thus not plotted in this figure.

The Setting for Link Prediction. For Livejournal, we follow the exactly same settings in Pytorch-BigGraph. For other three billion-scale graphs, we follow LightNE to set up the link prediction evaluation. We randomly excludes 0.00001% edges from the training graph for evaluation. When training SketchNE on these three graphs, the spectral propagation step is omitted due to memory cost and we set $d = 32$ except Hyperlink2012, where we use $d = 16$. We rank positive edges among randomly sampled corrupted edges to get the ranking metrics on the test set after training. We evaluate the link prediction task with four metrics—mean rank (MR), HITS@10, HITS@50, and AUC.

C. Experimental Results

Vertex Classification Results. We summarize the multi-label vertex classification performance in Fig. 2. In BlogCatalog, SketchNE achieves significantly better Micro-F1 and Macro-F1 than the second best method LightNE (by 3.5% on average). In YouTube [21], SketchNE show comparable performance to LightNE and GraphVite, while show significantly better results than others. In OAG [39], SketchNE achieves better performance than LightNE—the second best baseline on this data (Micro-F1 improved by 5.4% on average). In Friendster-small, and Friendster [38], SketchNE achieves the best performance among all baselines. To illustrate the effectiveness of SketchNE versus GNN methods, we test them on the BlogCatalog and YouTube datasets. The non-informative features with Xavier uniform distribution or Xavier normal distribution show almost the same performance and we retain the better results between the two distributions in Fig. 2. On BlogCatalog, SketchNE achieves significantly better Micro-F1 and Macro-F1 than all of them—DGI, GraphCL, GCC, and GraphSage. On YouTube, GraphSAGE and GCC can complete the training, while GraphCL and DGI fail due to the limitation of GPU memory size. A recent work [52] revealed that DGI cannot scale to large graphs. Because GraphCL is built on top of DGI, it also cannot scale to large graphs. For very large graphs, all the GNN baselines cannot finish the training with its original code implementation due to either the limitation of GPU memory size or unconvergence within a reasonable time. Specifically, SketchNE outperforms GraphSAGE on YouTube (28.9% improvement for Micro-F1 and 44.6% for Macro-F1 on average). SketchNE achieves significantly better performance than GCC on YouTube (71.5% improvement for Micro-F1 and 197.4% for Macro-F1 on average).

Overall, SketchNE has significantly better or comparable classification results compared to other methods. Compared to RandNE, FastRP and NRP, which omit element-wise function for scalability, SketchNE shows significantly better performance. It proves that the element-wise function is crucial for learning high quality embedding and that the method in Section III-A to factorize element-wise function of low rank matrix is practical. Overall, the vertex classification results illustrate the effectiveness superiority of SketchNE.

TABLE IV
LINK PREDICTION COMPARISON. LJ, CW, HL14 AND HL12 STANDS FOR LIVEJOURNAL, CLUEWEB, HYPERLINK2014 AND HYPERLINK2012, RESPECTIVELY

Datasets	Systems	MR ↓	HITS@10 ↑	HITS@50 ↑	AUC ↑
LJ	GCC	31.80	0.335	0.735	0.689
	GraphSAGE	14.22	0.655	0.916	0.860
	PBG	4.25	0.929	0.974	0.968
	GraphVite	3.06	0.962	0.982	**0.973**
	NetSMF	3.09	0.954	0.986	0.935
	RandNE	5.19	0.912	0.966	0.957
	FastRP	4.51	0.928	0.973	0.965
	NRP	2.98	0.949	0.993	0.934
	ProNE	3.31	0.950	0.982	0.932
	LightNE	2.13	**0.977**	0.993	0.945
	SketchNE	**2.10**	**0.977**	**0.994**	0.945
CW	LightNE	105.9	0.753	0.803	0.903
	SketchNE	**32.0**	**0.771**	**0.869**	**0.968**
HL14	LightNE	129.7	0.5	0.628	0.874
	SketchNE	**110.3**	**0.593**	**0.693**	**0.890**
HL12	LightNE	257.7	0.189	0.348	0.751
	SketchNE	**68.1**	**0.722**	**0.802**	**0.933**

TABLE V
EFFICIENCY COMPARISON AMONG SKETCHNE AND OTHER NETWORK EMBEDDING BASELINES

Metric	Datasets	RandNE	FastRP	NRP	PBG	GraphVite	NetSMF	ProNE	LightNE	**SketchNE**
Time	BlogCatalog	**0.5 s**	1.0 s	3.0 s	174.0 s	4.0 s	11.3 m	79.0 s	152.0 s	2.0 s
	YouTube	**12.0 s**	17.0 s	173.0 s	12.5 m	44.0 s	3.7 h	65.0 s	96.0 s	40.0 s
	Friendster-small	11.8 m	40.0 m	3.5 h	22.7 m	2.8 h	52 m	5.3 m	7.5 m	**5.2 m**
	Friendster	57.8 m	3.5 h	16.1 h	5.3 h	20.3 h	16.5 h	19.5 m	37.6 m	**16.0 m**
	OAG	33.7 m	3.5 h	11.4 h	20 h	1+day	22.4 h	**22.6 m**	1.5 h	1.1 h
	Livejournal	**9.0 m**	18.0 m	4.3 h	7.3 h	29.0 m	2.1 h	12.8 m	16.0 m	12.5 m
	ClueWeb	×	×	×	1+day	1+day	×	×	1.3 h	**37.7 m**
	Hyperlink2014	×	×	×	1+day	1+day	×	×	1.8 h[1]	**0.98 h**
	Hyperlink2012	×	×	×	1+day	1+day	×	×	5.6 h[1]	**1.0 h**
Mem (GB)	BlogCatalog	**0.2**	0.3	0.6	⋆	⋆	135	18	273	17
	YouTube	**6.9**	12.5	9.7	⋆	⋆	854	28	83	27
	Friendster-small	125	125	105	⋆	⋆	85	84	541	**56**
	Friendster	548	583	400	⋆	⋆	1144	326	559	**236**
	OAG	429	746	473	⋆	⋆	1500	403	1391	**283**
	Livejournal	224	417	282	⋆	⋆	140	**131**	532	147
	ClueWeb	×	×	×	⋆	⋆	×	×	1493	**612**
	Hyperlink2014	×	×	×	⋆	⋆	×	×	1500[1]	**1076**
	Hyperlink2012	×	×	×	⋆	⋆	×	×	1500[1]	**1321**

[1] LightNE requires sufficient samples that cost more than 1500GB mem (and more time), so it has to stop before it reaches the mem limit.
"⋆" indicates that we do not compare the memory cost of the CPU-GPU hybrid system (GraphVite) or distributed memory system (PBG).
"×" indicates that the corresponding algorithm is unable to handle the computation due to excessive space and memory consumption.

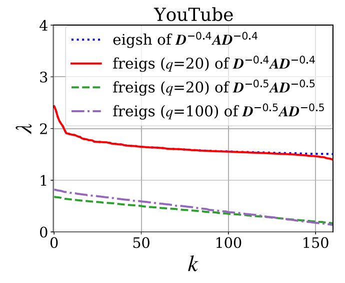
Link Prediction Results. Table IV lists the link prediction performance. For Livejournal, SketchNE outperforms all baselines in terms of MR, HITS@10, and HITS50. For the three billion-scale networks—ClueWeb, Hyperlink2014, and Hyperlink2012, we only report the results of LightNE and SketchNE, while other network embedding methods cannot finish running due to excessive memory or/and time cost, and GNN methods fail due to either the limitation of GPU memory size or unconvergence within a reasonable time. The results of LightNE are reported by choosing edge sample parameters to reach 1.5 TB memory bound. On these datasets, SketchNE produces significant outperformance over LightNE as measured by all four metrics. Take Hyperlink2012—the largest one with 3.5 billion vertices and 225 billion edges—for example, SketchNE achieves relative gains of 278%, 282%, 130%, and 24% over LightNE (the second best baseline on Livejournal) in terms of MR, HITS@10, HITS@50, and AUC.

More Discussion on the Performance of GNNs. It is interesting to see that network embedding (NE) methods (not only our SketchNE, but also other NE methods such as FastRP, NRP and ProNE) significantly outperform GNN methods in vertex classification on BlogCatalog and YouTube datasets, as well as link prediction on Livejournal dataset. We attribute the poor performance of GNNs to the nature of the three datasets themselves. These datasets (and corresponding tasks) are mainly proximity-based, so the inductive bias of NE methods that proximal vertices should have similar representations can significantly boost their performance. However, GNN methods usually consider more complex and general information such as structure and attributes, which may limit their performance in proximity-based tasks.

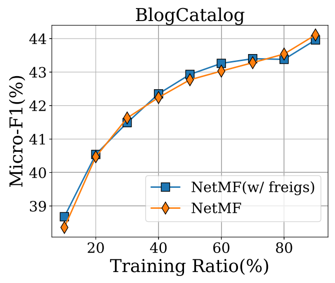
Time and Memory Efficiency. We report the running time and memory cost of SketchNE and other eight network embedding baselines on all nine datasets in Table V. Time-wise, on small datasets with millions of edges, the running time of SketchNE is relatively comparable to other baselines. However, on networks of billions of edges, e.g., Friendster, ClueWeb, Hyperlink2014 and Hyperlink2012, it takes SketchNE the least time to embed them. The GNN baselines, DGI and GraphCL are limited by GPU memory for the full-graph training and difficult to scale to large graphs, while GCC and GraphSAGE show slow convergence because it is based on graph sampling. For example, the training of GraphSAGE (trained via neighborhood sampling, a popular graph sampling method) on BlogCatalog (the smallest graph which consists of 10,312 vertices and 333,983 edges) using a GeForce GTX 1080 Ti GPU costs more than six hour, while it costs only 2 seconds for SketchNE to run the same task. We list all the time comparison among SketchNE and GNN baselines in Table VI. Memory-wise, we can observe that SketchNE demands less memory than all baselines on all datasets except using slightly more memory than NetSMF and ProNE on the small Livejournal data, empowering it to go for

TABLE VI
TIME COMPARISON AMONG SKETCHNE AND GNN BASELINES. "×" INDICATES THAT THE CORRESPONDING ALGORITHM IS UNABLE TO HANDLE THE COMPUTATION DUE TO THE LIMITED-SIZE GPU MEMORY. LARGER DATASETS ARE NOT LISTED DUE TO THE SCALABILITY ISSUE OF GNN MODELS

Baselines	BlogCatalog	YouTube	Livejournal
DGI	6.3 m	×	×
GraphCL	15.5 m	×	×
GCC	10.9 m	0.98 h	15.6 h
GraphSAGE	6.2 h	18.0 h	46.5 h
SketchNE	**2.0 s**	**40.0 s**	**12.5 m**



(a) The computed eigenvalues w.r.t. α



(b) NetMF vs. NetMF (w/ freigs)

Fig. 3. The validation of the effectiveness of freigs.

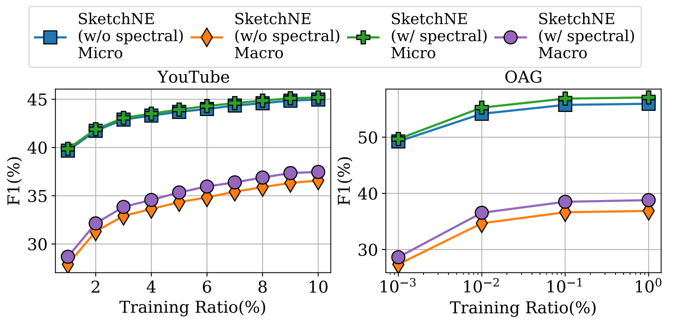


Fig. 4. The embedding performance comparison.

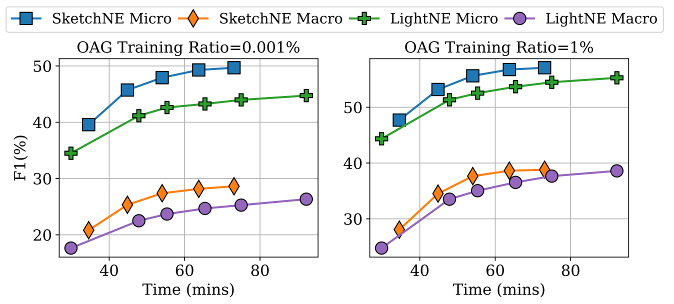


Fig. 5. The trade-offs between efficiency and performance.

the largest networks considered and beyond. For example, the running time of SketchNE on ClueWeb [42] is 37.7 minutes and the peak memory cost is 612 GB, which is 2× faster than LightNE and saves more than 59% memory. The results on Hyperlink2014 and Hyperlink2012 [43] further demonstrate the efficiency of SketchNE. It is worth noting that SketchNE can embed the Hyperlink2012 network with 3.5 billion vertices and 225 billion edges in 1.0 hours by using 1,321 GB memory on a single machine. In conclusion, the results on the three very large-scale graphs demonstrate that SketchNE can achieve consistently and significantly better efficiency than LightNE in terms of both running time and memory cost.

D. Ablation and Case Studies

Efficiency and Effectiveness of Each Step of SketchNE. First, we focus on the fast randomized eigen-decomposition (freigs, Algorithm 6 Line 1). Halko et al. [29] has shown that it is challenging to perform eigen-decomposition on $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$. We choose YouTube as an example, whose related eigenvalues are shown in Fig. 3(a). The eigenvalues of $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ with power iteration $q = 20$ and $q = 100$ are far from correct, as the correct largest eigenvalue should be 1. The eigsh [28] is not able to complete this job in three days. The results illustrate the requirement to use modified Laplacian matrix for fast randomized eigen-decomposition. When we choose $\alpha = 0.4$, the eigenvalues of fast randomized eigen-decomposition ($q = 20$) on $\boldsymbol{D}^{-0.4}\boldsymbol{A}\boldsymbol{D}^{-0.4}$ is indistinguishable from the eigenvalues computed by eigsh when $k < 140$. The running time of fast randomized eigen-decomposition is 20 seconds while eigsh costs 31 minutes, which proves the accuracy and efficiency of fast randomized eigen-decomposition. Then, we replace the eigen-decomposition (eigsh) of NetMF with Algorithm 5 and evaluate the Micro-F1 result of BlogCatalog between NetMF and NetMF (w/ freigs) in Fig. 3(b). The results show the effectiveness of freigs. Next, we focus on the effects of sparse-sign randomized single-pass SVD and spectral propagation. SketchNE (w/o spectral) and SketchNE (w/ spectral) represent the result of initial and enhanced embeddings, respectively. The vertex classification result is shown in Fig. 4. The performance of model (w/o spectral) has been satisfactory, which proves the effectiveness of the sparse-sign randomized single-pass SVD. Combining with spectral propagation, SketchNE (w/ spectral) shows better results, demonstrating the effect of spectral propagation.

The Effects of Parameter q. Here we need to pay attention to q, which determines the accuracy of the fast randomized eigen-decomposition. Thus, we make a trade-off between the quality of the learned embeddings and the overall running time. The peak memory cost is constant when we fix the oversampling parameters, column density z, eigen-decomposition rank k, and embedding dimension d. By fixing other parameters for OAG, we enumerate q from $\{10, 15, 20, 25, 30\}$, the efficiency-effectiveness trade-off of OAG is shown as Fig. 5. We also add LightNE results with different edge samples M from $\{1Tm, 5Tm, 7Tm, 10Tm, 13Tm, 17Tm\}$. The peak memory of SketchNE is still 283 GB while that of LightNE is 553 GB, 682 GB, 776 GB, 936 GB, 1118 GB and **1391 GB**, respectively. Fig. 5 shows that SketchNE can learn more effective embeddings than LightNE when the running time is constant with less memory cost. The experiment proves that users can adjust SketchNE flexibly according to time/memory budgets and performance requirements.

The Effects of Parameter α. We also analyze the influence of parameter α, which balances the accuracy of fast randomized eigen-decomposition on modified Laplacian matrix and the approximation error of (11). We select here YouTube and OAG as example datasets for the ablation study. We vary the parameter α from $\{0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$ and fix other parameters. The change of SketchNE's performance as the α varies can be shown in Fig. 6. From it we can see that when setting $\alpha = 0.5$, the eigen-decomposition on the Laplacian matrix has a bad accuracy

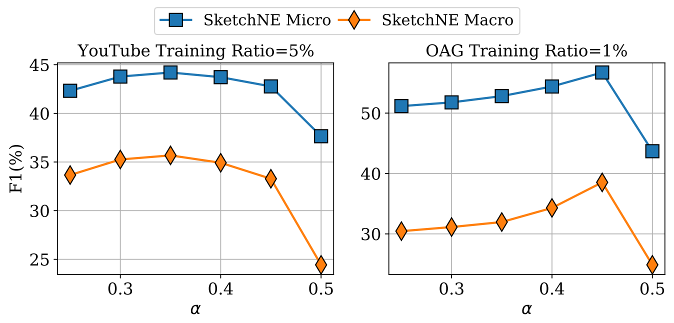


Fig. 6. The performance change when varying parameter α.

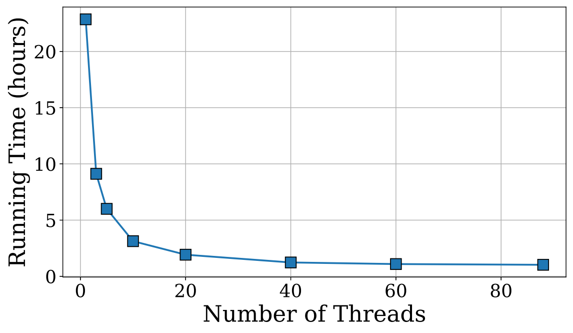


Fig. 7. Runtime of SketchNE v.s. the number of threads.

and therefore causes a loss of performance in Fig. 6. Fig. 6 also shows that SketchNE can learn almost the best embeddings when setting α in the range $[0.35, 0.45]$.

The Sketching Method. We download the official source codes[13] of Polynomial Tensor Sketch [31], and set $k = 10, r = 11, m = 12$, which makes dimension $m \times r + 1$ slightly bigger than SketchNE's $d = 128$ for BlogCatalog. We replace the sparse-sign randomized single-pass SVD in SketchNE with the polynomial tensor sketch algorithm, which is an alternative solution to factorize $f^{\circ}(\boldsymbol{LR})$. Experiments on BlogCatalog show that SketchNE with sparse-sign randomized single-pass SVD performs much better than SketchNE with polynomial tensor sketch, achieving a 44%/67.8% relative improvement on Micro-F1/Macro-F1 with 10% of training data.

The Number of Threads. In this work, we use a single-machine shared memory implementation with multi-threading acceleration. We set the number of threads to be 1, 3, 5, 10, 20, 40, 60, 88, and report the corresponding running time of SketchNE in Fig. 7. SketchNE takes 22.8 hours to embed the Hyperlink2012 network with 1 threads and 1.9 hours with 40 threads, achieving a $12\times$ speedup ratio (with ideal being $40\times$). This relatively good sub-linear speedup supports SketchNE to scale up to networks with hundreds of billions of edges.

V. Related Work

In this section, we first review the related work of network embedding and graph neural networks, and dicuss their difference from a vertex similarity perspective. Then we review related work on randomized matrix factorization.

Network Embedding (NE). Network embedding has been comprehensively studied over the past decade. Recent work about network embedding can be divided into two categories. The first category is based on skip-gram methods inspired by word2vec [10], including DeepWalk [2], Line [4], node2vec [20]. These methods rely on stochastic gradient descent to optimize a logistic loss. The second category method is based on matrix factorization, using SVD or other matrix decomposition techniques to generate the best low-rank approximation [53]. GraRep [54], HOPE [55], NetMF [13], NetSMF [7], ProNE [18], NRP [23], LightNE [17] and PANE [56] are methods in this category. There are several high performance embedding systems for large graphs have been developed. GraphVite [12], a CPU-GPU hybrid network embedding system, is developed based on DeepWalk [2] and LINE [4]. In GraphVite, CPU is used to perform graph operation, and GPU is used to compute linear algebra. Nevertheless, the GPU memory is a disadvantage when processing billion scale networks, limiting widespread use. Based on DeepWalk [2] and LINE [4], PyTorch-BigGraph [11] has been proposed for distributed memory machines. It achieves load balancing by graph partition and synchronization through a shared parameter server. In this work, we propose SketchNE, which leverages the merit of NetSMF and LightNE, and addresses their limitation in speed and memory overhead.

Graph Neural Networks (GNNs). GNNs introduce deep neural networks into graph learning, including GCN [57], GAT [58], GIN [59], ie-HGCN [48], GCC [46], GraphCL [45], DGI [44] and GraphSAGE [47]. In practice, scaling GNN models to large graphs can be time consuming. The training process of GNN models can be classified into two main types. The first type is full-graph training where each gradient descent step requires traversing the full graph and thus the time complexity is proportional to the graph size. The second type is graph sampling-based where each gradient descent step only involves sampled subgraphs. Although the time complexity for each gradient descent step is significantly reduced by sampling, the number of gradient descent steps required for convergence is empirically proportional to the graph size, which makes the overall time complexity proportional to the graph size. Therefore, training GNNs on billion-scale graphs relies on distributed computing supports with both CPUs and GPUs, such as the one used for the PinSage [9], Psgraph [60], AGL [61] and Neugraph [62] systems, while the premise of SketchNE is to embed billion-scale graphs into latent embeddings with only a CPU server efficiently.

In this work, we also compare several GNNs with our proposed network embedding solution, SketchNE. For a fair comparison with network embedding methods which mainly conduct unsupervised learning on graphs without attributes, we focus on GNNs which can be trained (1) in an unsupervised/self-supervised way and (2) without additional vertex and edge features, including GraphSAGE [47], DGI [44], GCC [46], GraphCL [45] and ie-HGCN [48]. GraphSAGE and DGI propose loss functions which can be used to train GNNs

[13]https://github.com/insuhan/polytensorsketch

models in an unsupervised manner. GCC proposes a framework to capture the universal network topological properties in a self-supervised way. GraphCL improves the performance of graph self-supervised learning by data augmentations. ie-HGCN replaces vertex attributes in GNNs with random initialized features.

Randomized Matrix Factorization. As the amount of data continues to increase, the popularity of randomized matrix computations has grown significantly. Randomized SVD can be an alternative to conventional SVD methods, because it involves the same or fewer floating-point operations and is more efficient for truly large high-dimensional data, by exploiting modern computing architectures [29]. As an application, frPCA [32] is developed for large-scale sparse data with better performance than conventional PCA algorithms. Randomized SVD has shown superiority in recent network embedding methods such as NetSMF, ProNE, and LightNE. Over the past few years, several single-pass SVD algorithms [30], [63] based on randomized matrix sketch have been introduced for streaming data scenarios. The efficiency and memory limitation of NetMF will be solved by randomized matrix factorization.

VI. Conclusion

In this work, we propose SketchNE, a fast, memory-efficient, and scalable network embedding method. We formulate the computation goal of NetMF as factorizing an element-wise function of low-rank matrix and then analyze its computational challenges. SketchNE resolves these challenges by leveraging various randomized linear sketch techniques, including but not limited to sparse-sign matrix, single-pass SVD, and fast eigen-decomposition. We use sparse-sign random projection matrix to solve the matrix multiplication challenge between $f^{\circ}(\boldsymbol{LR})$ and random projection matrix, and generate a sketch that can caputure the dominant information of $f^{\circ}(\boldsymbol{LR})$. Then, we solve the challenge in constructing reduced matrix by single-pass SVD. Second, we propose a fast randomized eigen-decomposition algorithm for modified Laplacian matrix. To enhance the performance of embedding, spectral propagation is adopted and a high-performance parallel graph processing stack GBBS is used to achieve memory-efficiency. The main computation steps of SketchNE are highly parallelizable, which is thus well supported by the MKL library and OpenMP. With the help of these techniques, SketchNE achieves the best performance on vertex classification and link prediction among state-of-the-art methods across diverse datasets. Notably, SketchNE can learn high-quality embeddings for a network with 3.5 billion vertices and 225 billion edges in 1.0 h by using 1,321 GB memory on a single machine, and the learned embeddings offer a 282% relative HITS@10 improvement over LightNE on the link prediction task. In the future, we plan to extend the method to handle dynamic and heterogeneous networks. Since the sparse-sign randomized single-pass SVD is proposed for solving the problem $f^{\circ}(\boldsymbol{LR})$, it is foreseeable that Algorithm 4 may have advantages in approximating the basic compoents in deep learning network.

References

[1] W. L. Hamilton, "Graph representation learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 14, no. 3, pp. 1–159, 2020.

[2] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 701–710.

[3] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale commodity embedding for e-commerce recommendation in Alibaba," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 839–848.

[4] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 1067–1077.

[5] R. Ramanath et al., "Towards deep and representation learning for talent search at linkedin," in *Proc. Conf. Inf. Knowl. Manage.*, 2018, pp. 2253–2261.

[6] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining.*, 2017, pp. 135–144.

[7] J. Qiu et al., "NetSMF: Large-scale network embedding as sparse matrix factorization," in *Proc. 24th Int. Conf. World Wide Web*, 2019, pp. 1509–1520.

[8] M. Academic, "Multi-sense network representation learning in microsoft academic graph," 2020. Accessed: Nov. 21, 2022. [Online]. Available: https://www.microsoft.com/en-us/research/project/academic/articles/multi-sense-network-representation-learning-in-microsoft-academic-graph/

[9] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 974–983.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. Int. Conf. Learn. Representations*, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[11] A. Lerer et al., "PyTorch-BigGraph: A large-scale graph embedding system," in *Proc. Mach. Learn. Syst.*, 2019, pp. 120–131.

[12] Z. Zhu, S. Xu, J. Tang, and M. Qu, "GraphVite: A high-performance CPU-GPU hybrid system for node embedding," in *Proc. 24th Int. Conf. World Wide Web*, 2019, pp. 2494–2504.

[13] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, LINE, PTE, and node2vec," in *Proc. 11th ACM Int. Conf. Web Search Data Mining*, 2018, pp. 459–467.

[14] A. Vaswani et al., "Attention is all you need," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[15] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.

[16] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *Ann. Statist.*, vol. 36, no. 3, pp. 1171–1220, 2008.

[17] J. Qiu, L. Dhulipala, J. Tang, R. Peng, and C. Wang, "LightNE: A lightweight graph processing system for network embedding," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 2281–2289.

[18] J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding, "ProNE: Fast and scalable network representation learning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 4278–4284.

[19] L. Dhulipala, G. E. Blelloch, and J. Shun, "Theoretically efficient parallel graph algorithms can be fast and scalable," in *Proc. Symp. Parallelism Algorithms Architectures*, 2018, pp. 393–404.

[20] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 855–864.

[21] L. Tang and H. Liu, "Scalable learning of collective behavior based on sparse social dimensions," in *Proc. Conf. Inf. Knowl. Manage.*, 2009, pp. 1107–1116.

[22] V. Ivashkin and P. Chebotarev, "Do logarithmic proximity measures outperform plain ones in graph clustering?," in *Proc. Int. Conf. Netw. Anal.*, 2016, pp. 87–105.

[23] R. Yang, J. Shi, X. Xiao, Y. Yang, and S. S. Bhowmick, "Homogeneous network embedding for massive graphs via reweighted personalized pagerank," *Proc. VLDB Endowment*, vol. 13, pp. 670–683, 2020.

[24] Z. Zhang, P. Cui, H. Li, X. Wang, and W. Zhu, "Billion-scale network embedding with iterative random projection," in *Proc. Int. Conf. Des. Mater.*, 2018, pp. 787–796.

[25] H. Chen, S. F. Sultan, Y. Tian, M. Chen, and S. Skiena, "Fast and accurate network embeddings via very sparse random projection," in *Proc. Conf. Inf. Knowl. Manage.*, 2019, pp. 399–408.

[26] D. Cohen-Steiner, W. Kong, C. Sohler, and G. Valiant, "Approximating the spectrum of a graph," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1263–1271.

[27] J. Liu, C. Wang, M. Danilevsky, and J. Han, "Large-scale spectral clustering on graphs," in *Proc. Int. Joint Conf. Artif. Intell.*, 2013, pp. 1486–1492.

[28] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *Arpack Users' Guide: Solution of Large-Scale Eigenvalue Problems With Implicitly Restarted Arnoldi Methods*. Philadelphia, PA, USA: SIAM, 1998.

[29] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011.

[30] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, "Streaming low-rank matrix approximation with an application to scientific simulation," *SIAM J. Sci. Comput.*, vol. 41, no. 4, pp. A2430–A2463, 2019.

[31] I. Han, H. Avron, and J. Shin, "Polynomial tensor sketch for element-wise function of low-rank matrix," in *Proc. IEEE Int. Conf. Mach. Learn. Appl.*, 2020, pp. 3984–3993.

[32] X. Feng, Y. Xie, M. Song, W. Yu, and J. Tang, "Fast randomized PCA for sparse data," in *Proc. Asian Conf. Mach. Learn.*, 2018, pp. 710–725.

[33] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," *ACM Sigplan Notices*, vol. 48, no. 8, 2013, pp. 135–146.

[34] J. Kepner and J. Gilbert, *Graph Algorithms in the Language of Linear Algebra*. Philadelphia, PA, USA: SIAM, 2011.

[35] J. Shun, L. Dhulipala, and G. E. Blelloch, "Smaller and faster: Parallel processing of compressed graphs with ligra+," in *Proc. Data Compression Conf.*, 2015, pp. 403–412.

[36] L. Dagum and R. Menon, "OpenMP: An industry-standard API for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, First Quarter 1998.

[37] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2009, pp. 817–826.

[38] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowl. Inf. Syst.*, vol. 42, no. 1, pp. 181–213, 2015.

[39] A. Sinha et al., "An overview of microsoft academic service (MAS) and applications," in *Proc. 24th Int. Conf. World Wide Web Companion*, 2015, pp. 243–246.

[40] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and mining of academic social networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2008, pp. 990–998.

[41] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Math.*, vol. 6, no. 1, pp. 29–123, 2009.

[42] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. 24th Int. Conf. World Wide Web*, 2004, pp. 595–601.

[43] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer, "The graph structure in the web – analyzed on different aggregation levels," *J. Web Sci.*, vol. 1, pp. 33–47, 2015.

[44] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rklz9iAcKQ

[45] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 5812–5823.

[46] J. Qiu et al., "GCC: Graph contrastive coding for graph neural network pre-training," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 1150–1160.

[47] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[48] Y. Yang, Z. Guan, J. Li, W. Zhao, J. Cui, and Q. Wang, "Interpretable and efficient heterogeneous graph convolutional network," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 2, pp. 1637–1650, Feb. 2023.

[49] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[50] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, 2008.

[51] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *Data Mining and Knowledge Discovery Handbook*. Berlin, Germany: Springer, 2009, pp. 667–685.

[52] Y. Zheng, S. Pan, V. C. Lee, Y. Zheng, and P. S. Yu, "Rethinking and scaling up graph contrastive learning: An extremely efficient approach with group discrimination," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 10809–10820.

[53] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[54] S. Cao, W. Lu, and Q. Xu, "GraRep: Learning graph representations with global structural information," in *Proc. Conf. Inf. Knowl. Manage.*, 2015, pp. 891–900.

[55] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1105–1114.

[56] R. Yang, J. Shi, X. Xiao, Y. Yang, J. Liu, and S. S. Bhowmick, "Scaling attributed network embedding to massive graphs," *Proc. VLDB Endowment*, vol. 14, no. 1, pp. 37–49, 2020.

[57] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[58] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[59] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[60] J. Jiang et al., "PSGraph: How tencent trains extremely large-scale graphs with spark?," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1549–1557.

[61] D. Zhang et al., "AGL: A scalable system for industrial-purpose graph machine learning," in *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3125–3137, 2020.

[62] L. Ma et al., "Neugraph: Parallel deep neural network computation on large graphs," in *Proc. USENIX Conf. Usenix Annu. Techn. Conf.*, 2019, pp. 443–457.

[63] W. Yu, Y. Gu, J. Li, S. Liu, and Y. Li, "Single-pass PCA of large high-dimensional data," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 3350–3356.

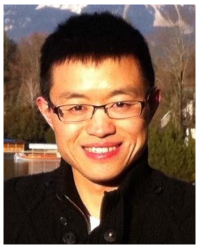
Yuyang Xie currently working toward the PhD degree in the Department of Computer Science and Technology, Tsinghua University. His main research interests include network embedding, representation learning and matrix computation.

Yuxiao Dong (Senior Member, IEEE) is an Assistant Professor of computer science with Tsinghua University. His research focuses on data mining, graph representation learning, foundation models, and social networks, with an emphasis on developing machine learning models to addressing problems in Web-scale systems. He received the 2017 SIGKDD Dissertation Award Honorable Mention, 2022 IJCAI Early Career Spotlight, and 2022 SIGKDD Rising Star Award.

Jiezhong Qiu received the PhD degree from Tsinghua University, in 2022. He is a principal investigator of Research Center for Intelligent Computing Platforms, Zhejiang Lab. His research interests include algorithm design for large-scale information networks and representation learning for graph-structured data. He received the 2022 SIGKDD Dissertation Award Runner-up.

Wenjian Yu (Senior Member, IEEE) received the BS and PhD degrees in computer science from Tsinghua University, Beijing, China, in 1999 and 2003, respectively. He is currently a professor with the Department of Computer Science and Technology with Tsinghua University. He has authored/coauthored three books and about 200 papers in refereed journals and conferences. His current research interests include high-performance numerical algorithms, big-data analytics, machine learning, and electronic design automation.

Xu Feng currently working toward the PhD degree in the Department of Computer Science and Technology, Tsinghua University. His main research interests include randomized matrix decomposition and data analysis for large-scale matrix.



Jie Tang (Fellow, IEEE) is a professor of the Department of Computer Science with Tsinghua University. He is a Fellow of the ACM and Fellow of AAAI. His interests include artificial general intelligence, data mining, social networks, and machine learning. He was honored with the SIGKDD Test-of-Time Award and SIGKDD Service Award.