Efficient Training of Graph Neural Networks on Large Graphs

Yanyan Shen
Shanghai Jiao Tong University
shenyy@sjtu.edu.cn

Lei Chen
HKUST, HKUST(GZ)
leichen@cse.ust.hk

Jingzhi Fang
HKUST
jfangak@connect.ust.hk

Xin Zhang
HKUST
xzhanggb@connect.ust.hk

Shihong Gao
HKUST
sgaoar@connect.ust.hk

Hongbo Yin
HKUST(GZ)
hyin744@connect.hkust-gz.edu.cn

ABSTRACT

Graph Neural Networks (GNNs) have gained significant popularity for learning representations of graph-structured data. Mainstream GNNs employ the message passing scheme that iteratively propagates information between connected nodes through edges. However, this scheme incurs high training costs, hindering the applicability of GNNs on large graphs. Recently, the database community has extensively researched effective solutions to facilitate efficient GNN training on massive graphs. In this tutorial, we provide a comprehensive overview of the GNN training process based on the graph data lifecycle, covering graph preprocessing, batch generation, data transfer, and model training stages. We discuss recent data management efforts aiming at accelerating individual stages or improving the overall training efficiency. Recognizing the distinct training issues associated with static and dynamic graphs, we first focus on efficient GNN training on static graphs, followed by an exploration of training GNNs on dynamic graphs. Finally, we suggest some potential research directions in this area. We believe this tutorial is valuable for researchers and practitioners to understand the bottleneck of GNN training and the advanced data management techniques to accelerate the training of different GNNs on massive graphs in diverse hardware settings.

PVLDB Reference Format:
Yanyan Shen, Lei Chen, Jingzhi Fang, Xin Zhang, Shihong Gao, and Hongbo Yin. Efficient Training of Graph Neural Networks on Large Graphs. PVLDB, 17(12): 4237 - 4240, 2024.
doi:10.14778/3685800.3685844

1 INTRODUCTION

Graph Neural Networks (GNNs) have emerged as a powerful tool for extracting complex structural and semantic information from massive graphs. They have gained continuous popularity in a broad range of graph-based applications including social network analysis [15], financial risk assessment [2], knowledge graph construction [35], recommendation systems [10], etc.

Despite the variation in model architectures, most GNNs adopt the message passing scheme that exchanges information among nodes based on the graph structure. This scheme involves recursive neighbor information aggregation, which poses significant challenges when training GNNs on large graph datasets. These challenges include (1) high communication cost among machines for full-batch GNN training, (2) excessive data movement between CPU and GPU for mini-batch GNN training, and (3) low hardware utilization during model training. As highlighted by Sancus [27], when performing full-batch training of the GCN model [20] on the Ogbn-Products [14] dataset with an eight-GPU machine, the communication time accounts for over 90% of the end-to-end training time, even for highly optimized GNN training systems. Similarly, data movement dominates the end-to-end mini-batch GNN training time. The situation worsens for larger graph datasets like Ogbn-Papers100M [14]. Moreover, a recent study [1] has revealed that better computation resource utilization can lead to approximately 10× speedup in per-epoch GNN training efficiency.

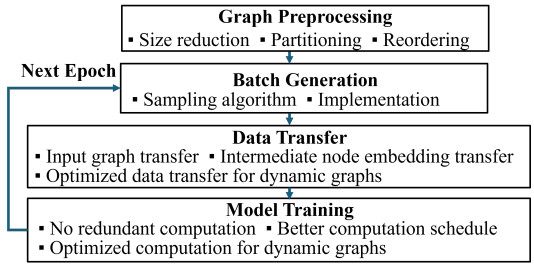

Figure 1: GNN training workflow.

As graph data continues to grow in size, developing efficient GNN training systems becomes increasingly crucial. At a high level, current training systems process graph data on hybrid CPU-GPU platforms, following the intricate computational logic of forward and backward propagation and facilitating data movement across heterogeneous hardware. With a longstanding history of studying efficient methods for complex graph processing, database researchers have recently directed their efforts toward exploring data management techniques to accelerate the GNN training process. This line of research generally progresses in parallel with GNN model development but emphasizes achieving better training efficiency without compromising model performance.

In this tutorial, we aim at providing a comprehensive overview of the recent efforts for accelerating GNN training on massive graphs. Based on the graph data lifecycle, we view the training process through the lens of four major stages: graph preprocessing, batch generation, data transfer, and model training. We discuss

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685844

innovative data management ideas at different stages that improve GNN training efficiency. Both centralized and distributed training environments will be considered. Due to the distinct characteristics of static and dynamic graphs and their respective GNN models, we first focus on efficient GNN training on large static graphs. Subsequently, we will delve into the efforts aimed at training GNNs on large dynamic graphs.

Tutorial outline and target audience. We plan for a 1.5-hour tutorial. The tutorial is intended for researchers and practitioners interested in GNN training systems, data management techniques aimed at improving efficiency and scalability within these training systems, as well as the broader field of scalable graph learning. We organize the tutorial into the following sections.

- **Introduction** (15 mins): We provide an overview of key concepts related to graphs, GNNs, and the systems tailored for GNN training.
- **Graph preprocessing** (10 mins): We introduce three graph preprocessing methods that can enhance the efficiency of subsequent GNN training processes.
- **Batch generation** (15 mins): We focus on the sampling methods used during batch generation, which are efficient and effective across different hardware settings.
- **Data transfer** (15 mins): We discuss efficient input graph data and intermediate embedding transfer methods between CPU and GPU, as well as among GPUs.
- **Model training** (10 mins): We present advanced techniques that reduce the computation cost of the training workloads and improve the utilization of computation resources.
- **Training on dynamic graphs** (15 mins): We cover the existing methods to accelerate GNN training on dynamic graphs.
- **Future research directions** (10 mins): We conclude the tutorial by exploring the potential future research directions on data management for efficient GNN training.

Related tutorials. There is a tutorial about large-scale GNNs [31], which focuses on advanced algorithms for training GNNs at scale. Our tutorial has a stronger emphasis on exploring data management techniques and system-level optimizations that can alleviate the efficiency bottlenecks of GNN training. It covers different hardware settings and GNNs for both static and dynamic graphs.

2 GRAPH PREPROCESSING

To enhance the efficiency of GNN training, existing works propose to perform necessary graph preprocessing. In this section, we introduce three common preprocessing techniques, namely graph size reduction, graph partitioning, and graph reordering, and discuss how they influence the later batching and transferring efficiency of the GNN training.

Graph size reduction. Training on large graphs can be time-consuming, and a straightforward idea to address this problem is to compact the training graph while maintaining the model accuracy. To do this, three common techniques are available: graph coarsening, graph sparsification, and graph condensation. Graph coarsening [24] tries to group nodes in the original graph into supernodes so that the graph size is reduced. Graph sparsification [19] removes the edges in the original graph to eliminate redundant and insignificant relationships. Graph condensation is different from the above two techniques in that it tries to construct synthetic smaller graphs. The graph condensation problem is often formulated as a bi-level optimization problem: we update the synthetic data in the outer loop based on the neural network trained on the synthetic data in the inner loop. Solving the bi-level optimization problem can be computationally expensive or even intractable. To improve the graph condensation efficiency, existing works try to (1) speed up the inner loop [6], or (2) avoid the bi-level optimization by solving the inner loop model training problem with a closed-form solution [30]. There are also works about graph-free condensation for better generalization ability [37].

Graph partitioning. Due to the limited device memory, graph partitioning on graph structure and feature data is employed to create smaller subgraphs for distributed GNN training. The current partitioning algorithms aim to reduce the communication cost and balance the workloads across different subgraphs. To partition the graph structure, we can use traditional graph partitioning algorithms [4, 27] (e.g., METIS partitioning, random partitioning) or strategies tailored for GNN training [17, 26]. For feature data, P^3 [9] optimize the communication cost by vertically partitioning.

Graph reordering. Reordering the vertices in graph is an effective way to enhance locality, which benefits both model computation and data transfer in GNN training. For example, GNNAdvisor [29] reorders the graph selectively to improve the locality of computation in graph convolution kernels. DUCATI [34] also proposes to reorder the graph based on access frequency to facilitate a lightweight cache construction and lookup for the topology data of the graph dataset. When traversing the graph topology with the unified memory, the HALO reordering [13] can improve the utilization of PCIe bandwidth and accelerate batch data transfer.

3 BATCH GENERATION

In GNN training, graph sampling has emerged as a powerful technique for generating batches. It selects nodes and their associated edges from the original graph. The sampled subgraphs serve as the batches that are fed into the downstream GNN training.

Graph sampling algorithms. A variety of graph sampling approaches have been tailored specifically to the characteristics of GNN training workflows. Node-wise sampling aggregates messages from a subset of neighbors, mitigating the computational challenges arising from the exponentially increasing dependencies in GNNs. Layer-wise sampling employs independent sampling for each network layer and leverages importance sampling to maintain a constant sample size, effectively controlling the computational growth. Subgraph-wise sampling generates a subgraph and reuses it for the computation across all layers. Furthermore, certain studies have highlighted the benefits of delicately considering the spatial locality in feature storage and extraction, which can significantly reduce the transfer cost. Feature-oriented sampling [33] initiates the sampling process by selecting node features, which then guides the formation of the corresponding subgraph for a mini-batch.

Graph sampling implementations. Efficient sampling involves leveraging the computational capabilities of available hardware resources in the system. The GPU serves as an ideal hardware choice for this purpose due to its excellent parallel computing power. Meanwhile, generating the sampled subgraphs on the GPU avoids the additional costs of transferring graph samples into the device

memory. Nextdoor [16] is a pioneer work that performs graph sampling on a single GPU. It loads the entire graph into device memory and proposes a transit-parallelism strategy to accelerate the sampling process by aggregating the same transit nodes. To enhance scalability through the extensive capacity of host memory, the Unified Virtual Addressing (UVA) technique is adapted for graph sampling [4, 34]. It stores the graph structure in host memory and executes graph sampling on GPUs, taking advantage of the combined resources of both the host and the device.

4 DATA TRANSFER

Efficient data transfer between CPU and GPU(s), or among multiple GPUs, is paramount for ensuring high GNN training efficiency on large-scale graphs. This is because the required data for model computation may not be locally available. The specific data that needs to be transferred depends on the applied GNN training approach, i.e., mini-batch based or full-batch based GNN training.

Transferring input graph data. For mini-batch based GNN training on large-scale graphs, cross-device transfer of the input graph data (initial node feature vectors and graph structure information) is a necessity. However, this data transfer process incurs substantial costs during training. To alleviate this issue, a common practice is to utilize local graph data caches to reduce the volume of data being transferred and employ sophisticated cache management policies. PaGraph [26] and GNNLab [32] exploit node feature locality, caching node feature vectors locally prior to training with certain hotness metrics. DUCATI [34] further exploits the locality of graph structure data and introduces an additional graph structure data GPU cache. It also leverages a refined allocator for managing both types of caches, to minimize the aggregate transfer volume of both types of graph input data between CPUs and GPUs.

Transferring intermediate node embeddings. For full-batch based GNN training, a large graph is typically partitioned across devices. In this setup, intermediate node embeddings, capturing the neighborhood-aggregated node representations across different GNN layers, need to be transferred across devices to enable full-neighbor aggregation of target nodes. DGCL [3] finds optimal transfer routes for every node's intermediate embeddings given the system topology to minimizing the corresponding transfer costs. Neutronstar [28] achieves reduced transfer cost for intermediate node embeddings at the expense of redundant bottom-up neighborhood aggregation locally. Sancus [27] proposes a skip-broadcast data transfer mechanism that flexibly bypasses intermediate node embedding transfer between GPUs during specific training epochs. This is guided by several refined management regarding the staleness of the embeddings.

5 MODEL TRAINING

The efficiency of model training (i.e., the forward and backward propagation) is also important for the overall training efficiency. Researchers have explored approaches to avoid redundant computation [18] or optimize the computation schedule, i.e., optimizing resource allocation of different operations for better resource utilization in a collaborative CPU-GPU computation setup [25], optimizing training parallelism [25], and improving the efficiency of computation graphs and operators on hardware [7, 8].

6 TRAINING ON DYNAMIC GRAPHS

Apart from the traditional static graphs, many real-world graphs exhibit dynamic behaviors with evolving interaction patterns. Typically, there are two types of dynamic graphs: discrete-time dynamic graphs (DTDGs) and continuous-time dynamic graphs (CTDGs), which are typically handled by dynamic GNNs (D-GNNs) and temporal GNNs (T-GNNs), respectively. The dynamic nature of these graphs introduces new challenges and optimizations for accelerating data transfer and computation during GNN training.

Optimized data transfer for dynamic graphs. For D-GNN training, Chakaravarthy *et al.* [5] capitalize the topological similarities across different graph snapshots, and design a graph-difference based transfer method to reduce the transfer cost of graph snapshots. For T-GNN training, ETC [11] identifies the redundant data access patterns and utilizes a redundancy-aware data access policy to reduce the transfer volume of input features. SIMPLE [12] further captures the complex entanglement effect between time and frequency information in CTDGs, and reduces the data transfer cost via dynamic data placement on GPU.

Optimized computation for dynamic graphs. Li *et al.* [21] identify the reusable intermediate embeddings across graph snapshots in D-GNN training, and propose cache policies that maximize the saved model computation time for the next snapshot. Orca [22] also caches intermediate embedding in T-GNN training. It employs a reuse-distance based cache policy, which optimally minimizes the overall re-computation costs under the given GPU cache budget. Zebra [23] points out the temporal effect of node influence in T-GNN training. It performs single-layer aggregation over the most influential neighbors, improving computational efficiency.

7 FUTURE WORK

Considering more tiers of memory hierarchy. There exist extremely large real-world graphs that exceed the total main memory capacity of multiple machines. In this case, we need to leverage additional tiers of the memory hierarchy, e.g., SSD, HDD, distributed storage, and cloud storage services, to train GNNs. Training GNNs efficiently in this situation demands future research efforts since the preferred access pattern and access latency of these memory levels are considerably different from that of the main memory.

System evolution for dynamic graphs. Dynamic graphs exhibit evolving structures and features over time, which poses additional challenges for continuous GNN training. The data access and data transfer patterns, which form the basis for the optimizations in batch generation and data transfer, will gradually change after the initial system deployment. To sustain the efficiency of GNN training systems, future work needs to devise adjustable and evolving systems as the underlying graph data changes.

Adaptation to enhanced GNN models on TAGs. Conventional GNNs can only process numerical vertex features, which are often lossy compressions of the original rich textual features in the text-attributed graphs (TAGs). Recently, there has been increasing interest in modifying the architecture of GNNs to improve their learning capability on TAGs [36]. However, the newly introduced components, such as word embedding layers, are positioned differently against the original GNN layers and cause a significant drift

of workload distribution. Therefore, adjusting system designs for enhanced GNN models on TAGs is also a potential future direction.

8 PRESENTERS

(1) **Yanyan Shen** is an associate professor at Shanghai Jiao Tong University. She received her Ph.D. degree in computer science from National University of Singapore in 2015. Her research interests lie in data management for machine learning algorithms and systems.
(2) **Lei Chen** is a chair professor of the Data Science and Analytic Thrust at HKUST (GZ). His research interests include data-driven AI and query optimization on large graphs. He received his Ph.D. degree in computer science from the University of Waterloo.
(3) **Jingzhi Fang** is a Ph.D. candidate in Computer Science and Engineering at HKUST. She is interested in accelerating the training and inference of AI models.
(4) **Xin Zhang** is a Ph.D. student in Data Science and Analytics at HKUST. He is interested in machine learning systems and GNNs.
(5) **Shihong Gao** is a Ph.D. student in Data Science and Analytics at HKUST with broad research interests in efficient machine learning.
(6) **Hongbo Yin** is a Ph.D. student in Data Science and Analytics at HKUST(GZ) with research interests in efficient graph learning.

ACKNOWLEDGMENTS

Yanyan Shen is supported by the National Key Research and Development Program of China (2022YFE0200500) and the Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102). Lei Chen is supported by National Key Research and Development Program of China Grant (2023YFF0725100), National Science Foundation of China (U22B2060), Guangdong Province Science and Technology Plan Project 2023A0505030011, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX.

REFERENCES

[1] Xin Ai, Qiange Wang, Chunyu Cao, Yanfeng Zhang, Chaoyi Chen, Hao Yuan, Yu Gu, and Ge Yu. 2023. NeutronOrch: Rethinking Sample-based GNN Training under CPU-GPU Heterogeneous Environments. *arXiv:2311.13225* (2023).
[2] Wendong Bi, Bingbing Xu, Xiaoqian Sun, Zidong Wang, Huawei Shen, and Xueqi Cheng. 2022. Company-as-tribe: Company financial risk assessment on tribe-style graph with hierarchical graph neural networks. In *KDD*. 2712–2720.
[3] Zhenkun Cai, Xiao Yan, Yidi Wu, Kaihao Ma, James Cheng, and Fan Yu. 2021. DGCL: An efficient communication library for distributed GNN training. In *EuroSys*. 130–144.
[4] Zhenkun Cai, Qihui Zhou, Xiao Yan, Da Zheng, Xiang Song, Chenguang Zheng, James Cheng, and George Karypis. 2023. DSP: Efficient GNN training with multiple GPUs. In *PPoPP*. 392–404.
[5] Venkatesan T Chakaravarthy, Shivmaran S Pandian, Saurabh Raje, Yogish Sabharwal, Toyotaro Suzumura, and Shashanka Ubaru. 2021. Efficient scaling of dynamic graph neural networks. In *SC*. 1–15.
[6] Junfeng Fang, Xinglin Li, Yongduo Sui, Yuan Gao, Guibin Zhang, Kun Wang, Xiang Wang, and Xiangnan He. 2024. Exgc: Bridging efficiency and explainability in graph condensation. *arXiv preprint arXiv:2402.05962* (2024).
[7] Jingzhi Fang, Yanyan Shen, Yue Wang, and Lei Chen. 2020. Optimizing DNN computation graph using graph substitutions. *VLDB* 13, 12 (2020), 2734–2746.
[8] Jingzhi Fang, Yanyan Shen, Yue Wang, and Lei Chen. 2024. STile: Searching Hybrid Sparse Formats for Sparse Deep Learning Operators Automatically. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–26.
[9] Swapnil Gandhi and Anand Padmanabha Iyer. 2021. P3: Distributed deep graph learning at scale. In *OSDI*. 551–568.
[10] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. 2022. Graph neural networks for recommender system. In *WSDM*. 1623–1625.
[11] Shihong Gao, Yiming Li, Yanyan Shen, Yingxia Shao, and Lei Chen. 2024. ETC: Efficient Training of Temporal Graph Neural Networks over Large-scale Dynamic Graphs. *VLDB* 17, 5 (2024), 1060–1072.
[12] Shihong Gao, Yiming Li, Xin Zhang, Yanyan Shen, Yingxia Shao, and Lei Chen. 2024. SIMPLE: Efficient Temporal Graph Neural Network Training at Scale with Dynamic Data Placement. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–25.
[13] Prasun Gera, Hyojong Kim, Piyush Sao, Hyesoon Kim, and David Bader. 2020. Traversing large graphs on GPUs with unified memory. *VLDB* 13, 7 (2020), 1119–1133.
[14] 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. *NeurIPS* 33 (2020), 22118–22133.
[15] Lokesh Jain, Rahul Katarya, and Shelly Sachdeva. 2023. Opinion leaders for information diffusion using graph neural network in online social networks. *TWEB* 17, 2 (2023), 1–37.
[16] Abhinav Jangda, Sandeep Polisetty, Arjun Guha, and Marco Serafini. 2021. Accelerating graph sampling for graph machine learning using GPUs. In *EuroSys*. 311–326.
[17] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the accuracy, scalability, and performance of graph neural networks with roc. *MLSys* 2 (2020), 187–198.
[18] Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. 2020. Redundancy-free computation for graph neural networks. In *KDD*. 997–1005.
[19] Xunqiang Jiang, Tianrui Jia, Yuan Fang, Chuan Shi, Zhe Lin, and Hui Wang. 2021. Pre-training on large-scale heterogeneous graph. In *KDD*. 756–766.
[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[21] Haoyang Li and Lei Chen. 2021. Cache-based gnn system for dynamic graphs. In *CIKM*. 937–946.
[22] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Orca: Scalable Temporal Graph Neural Network Training with Theoretical Guarantees. *Proc. ACM Manag. Data* 1, 1 (2023), 52:1–52:27. https://doi.org/10.1145/3588737
[23] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. *VLDB* 16, 6 (2023), 1332–1345.
[24] Zhiyuan Li, Xun Jian, Yue Wang, and Lei Chen. 2022. Cc-gnn: A community and contraction-based graph neural network. In *ICDM*. IEEE, 231–240.
[25] Zhiyuan Li, Xun Jian, Yue Wang, Yingxia Shao, and Lei Chen. 2024. DAHA: Accelerating GNN Training with Data and Hardware Aware Execution Planning. *VLDB* 17, 6 (2024), 1364–1376. https://www.vldb.org/pvldb/vol17/p1364-li.pdf
[26] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. 2020. Pagraph: Scaling gnn training on large graphs via computation-aware caching. In *SoCC*. 401–415.
[27] Jingshu Peng, Zhao Chen, Yingxia Shao, Yanyan Shen, Lei Chen, and Jiannong Cao. 2022. SANCUS: Staleness-Aware Communication-Avoiding Full-Graph Decentralized Training in Large-Scale Graph Neural Networks. *VLDB* 15, 9 (2022), 1937–1950. https://doi.org/10.14778/3538598.3538614
[28] Qiange Wang, Yanfeng Zhang, Hao Wang, Chaoyi Chen, Xiaodong Zhang, and Ge Yu. 2022. NeutronStar: Distributed GNN Training with Hybrid Dependency Management. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 1301–1315. https://doi.org/10.1145/3514221.3526134
[29] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. 2021. GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration on GPUs. In *OSDI*. 515–531.
[30] Zhe Xu, Yuzhong Chen, Menghai Pan, Huiyuan Chen, Mahashweta Das, Hao Yang, and Hanghang Tong. 2023. Kernel Ridge Regression-Based Graph Dataset Distillation. In *KDD*. 2850–2861.
[31] Rui Xue, Haoyu Han, Tong Zhao, Neil Shah, Jiliang Tang, and Xiaorui Liu. 2023. Large-Scale Graph Neural Networks: The Past and New Frontiers. In *KDD*. 5835–5836.
[32] Jianbang Yang, Dahai Tang, Xiaoniu Song, Lei Wang, Qiang Yin, Rong Chen, Wenyuan Yu, and Jingren Zhou. 2022. GNNLab: a factored system for sample-based GNN training over GPUs. In *EuroSys*. 417–434.
[33] Xin Zhang, Yanyan Shen, and Lei Chen. 2022. Feature-Oriented Sampling for Fast and Scalable GNN Training. In *ICDM*. IEEE, 723–732.
[34] Xin Zhang, Yanyan Shen, Yingxia Shao, and Lei Chen. 2023. DUCATI: A Dual-Cache Training System for Graph Neural Networks on Giant Graphs with the GPU. *Proc. ACM Manag. Data* 1, 2 (2023), 166:1–166:24. https://doi.org/10.1145/3589311
[35] Yongqi Zhang, Quanming Yao, Yingxia Shao, and Lei Chen. 2019. NSCaching: simple and efficient negative sampling for knowledge graph embedding. In *ICDE*. IEEE, 614–625.
[36] Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. 2022. Learning on large-scale text-attributed graphs via variational inference. *arXiv preprint arXiv:2210.14709* (2022).
[37] Xin Zheng, Miao Zhang, Chunyang Chen, Quoc Viet Hung Nguyen, Xingquan Zhu, and Shirui Pan. 2024. Structure-free graph condensation: From large-scale graphs to condensed graph-free data. *NeurIPS* 36 (2024).