Systems for Scalable Graph Analytics and Machine Learning: Trends and Methods

Da Yan
Indiana University Bloomington
yanda@iu.edu

Lyuheng Yuan
Indiana University Bloomington
lyyuan@iu.edu

Akhlaque Ahmad
Indiana University Bloomington
akahmad@iu.edu

Saugat Adhikari
Indiana University Bloomington
adhiksa@iu.edu

ABSTRACT

Graph-theoretic algorithms and graph machine learning models are essential tools for addressing many real-life problems, such as social network analysis and bioinformatics. To support large-scale graph analytics, graph-parallel systems have been actively developed for over one decade, such as Google's Pregel and Spark's GraphX, which (i) promote a think-like-a-vertex computing model and target (ii) iterative algorithms and (iii) those problems that output a value for each vertex. However, this model is too restricted for supporting the rich set of heterogeneous operations for graph analytics and machine learning that many real applications demand.

In recent years, two new trends emerge in graph-parallel systems research: (1) a novel think-like-a-task computing model that can efficiently support the various computationally expensive problems of subgraph search; and (2) scalable systems for learning graph neural networks. These systems effectively complement the diversity needs of graph-parallel tools that can flexibly work together in a comprehensive graph processing pipeline for real applications, with the capability of capturing structural features. This tutorial will provide an effective categorization of the recent systems in these two directions based on their computing models and adopted techniques, and will review the key design ideas of these systems. Slides are available at https://github.com/akhlaqueak/VLDB-2025-Tutorial.

PVLDB Reference Format:
Da Yan, Lyuheng Yuan, Akhlaque Ahmad, and Saugat Adhikari. Systems for Scalable Graph Analytics and Machine Learning: Trends and Methods. PVLDB, 18(12): 5460 - 5465, 2025.
doi:10.14778/3750601.3750695

1 INTRODUCTION

Background and Motivation. Pioneered by Google's Pregel, a lot of graph-parallel systems have been developed that adopt a think-like-a-vertex (TLAV) programming model and iterative computation model. However, TLAV systems are dedicated to scaling those graph problems that output a value for each vertex, such as random walks and graph traversal, while many real problems care about subgraph structures, such as finding functional groups in bioinformatics, and finding social communities [14].

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750695

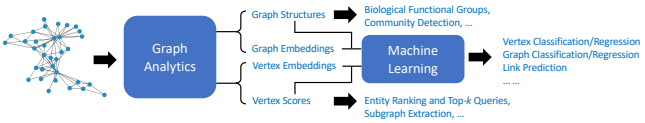

Figure 1: Pipeline for Graph Analytics and Learning

Figure 1 summarizes a typical pipeline for graph processing, consisting of a graph analytics phase and an optional graph machine learning (ML) phase. The analytic tasks either concern individual vertices (e.g., node scoring or classification), or concern substructures or even an entire graph (e.g., dense/frequent subgraph mining, graph classification). There are four analytics paths in the pipeline:

(1) **Vertex Analytics**, which outputs a score for each vertex, useful for applications such as biomolecule prioritization in network biology, or object ranking in recommender systems.
(2) **Vertex Analytics + ML**, where the analytics stage outputs vertex embeddings for downstream ML tasks. Vertex embeddings can be learned from the graph topology as in DeepWalk and node2vec, or the vertex features may come directly from the applications or be computed based on the graph topology (e.g., in- and out-degrees, clustering coefficient).
(3) **Structure Analytics**, which outputs subgraph structures (patterns or instances), useful for finding functional groups in network biology, and community detection.
(4) **Structure Analytics + ML**, where informative structures are extracted as features for graph classification/regression.

TLAV systems mainly address the scalability issue of vertex analytics (+ ML), with many killer applications in recommender systems and bioinformatics. However, many real problems concern subgraph structures, and they are actually more computationally challenging due to the exponential search space of subgraphs in a graph, but cannot be effectively accelerated by TLAV systems. For example, Chu and Cheng [9] noticed that for triangle counting, the state-of-the-art MapReduce algorithm takes 5.33 minutes using 1636 machines, while their serial external-memory algorithm takes only 0.5 minute. In fact, given an input graph $G = (V, E)$, TLAV systems are only efficient for iterative computations where each iteration has $O(|V| + |E|)$ cost and there are $O(\log |V|)$ iterations, giving a time complexity upper bound of $O((|V| + |E|) \log |V|)$ [52].

In recent years, two new trends emerge in graph-parallel systems research: (1) a lot of novel systems have been recently developed targeting the more compute-intensive subgraph search problems, which all adopt a subgraph-centric programming model (in contrast to vertex-centric); (2) graph neural networks (GNN) have boomed in various applications, and a number of scalable GNN systems

have been developed. These two directions well cover the "Graph Structures" and "ML" components of the pipeline shown in Figure 1, but there currently lacks a comprehensive tutorial to survey and introduce these exciting new system advancements.

This tutorial aims to fill this gap. We offered a tutorial on graph-parallel systems in SIGMOD 2016 [47] but it mainly focused on TLAV systems. There were tutorials on GNNs [2] that focused on model design in real applications rather than GNN system design. There was also a tutorial on training GNNs [32] in VLDB 2024 but the focus is more on algorithmic techniques as well as the handling of dynamic and temporal GNNs, while our tutorial component on GNN systems will also cover other system-related techniques such as how to effectively utilize new hardware and cloud platforms (e.g., NVLink, serverless computing) as well as more recent techniques such as utilizing lossy quantization for message compression. Besides, the other important and timely component on systems for structure analytics is new and not covered by those tutorials.

We remark that the two topics we cover here are related and important in order to fully explore the potential of various graph analytics tools in a real application pipeline. For example, frequent subgraph structural patterns have been found informative in conventional models for graph classification and regression [28, 31]. ML applications benefiting from having structural features include biochemistry [28], bioinformatics [29], and community detection [35], where structural features are found to outperform neural graph embedding methods. There are also works applying GNNs for approximate subgraph search, such as neural subgraph matching [61] and neural subgraph counting [40], where considering subgraph structures were found essential for good performance. Finally, Subgraph GNNs [5, 12] which model graphs as collections of subgraphs are found to be more expressive than regular GNNs.

2 SYSTEMS FOR STRUCTURE ANALYTICS

Programmability is important for a graph-parallel system: the system should make it easy to implement a broad range of advanced parallel/distributed analytics, not much more difficult (if not easier) than their serial algorithm counterparts. TLAV systems are a good example, where the user-specified programs are often easier to implement than a serial algorithm from scratch.

However, TLAV systems are not suitable for subgraph search, since computations are at individual vertices rather than subgraphs, and TLAV systems are for iterative computations with a time complexity of $O((|V| + |E|) \log |V|)$ [52].

In this tutorial, we will review a series of new systems proposed recently for subgraph search, which is a broad topic covering many problems such as subgraph enumeration/matching/counting, maximal/maximum clique finding, and **frequent subgraph pattern mining (FSM)**. Notably, FSM is different from the other problems, since FSM summarizes pattern graphs from the data graph(s), while all the other problems find valid subgraph instances of a data graph, which we call collectively as **subgraph finding (SF)**.

Different from TLAV systems, the systems for subgraph search adopt a think-like-a-graph (TLAG) computing model, which extends valid small graph structures by one edge (or vertex) at a time to grow larger valid graph structures. However, in order to support SF and FSM under a unified programming model, most of these systems such as Arabesque [38], RStream [41] and Pangolin [8] adopt a breadth-first subgraph extension approach where subgraphs of size $(i + 1)$ cannot start their generation until all subgraphs with size i have been generated, which creates a lot of subgraph materialization cost and restricts scalability since the number of subgraph instances grows exponentially with the input graph size.

Table 1: Systems for Subgraph Search: Summary of Features

	Problem Type: (Pattern-to-Instance) Subgraph Finding	Problem Type: (Pattern-to-Instance) FSM (A Big Graph)	Problem Type: (Instance-to-Pattern) FSM (transaction database)	Problem Type: (Instance-to-Pattern) Subgraph Materialization	Search Approach: Sugraph Backtracking	Search Approach: GPU Support
* SE/SM = Subgraph Enumeration/Matching * EG = Entire Graph (Loaded to GPU) * PT = A Partition of a Graph (Loaded to GPU) Each Time						
Arabesque	✓	✓		✓		
RStream	✓	✓		✓		
Pangolin	✓	✓		✓		EG
Peregrine	✓	✓		✓		
Fractal	✓	✓			✓	
AutoMine	✓	✓			✓	
G-thinker	✓				✓	
G-thinkerQ						
G-Miner	✓				✓	
GraphPi	SE/SM				✓	
GraphZero	SE/SM				✓	
T-DFS	SE/SM				✓	EG
GSI, cuTS, STMatch, EGSM	SE/SM			✓		EG
PBE, VSGM, SGSI	SE/SM			✓		PT
G^2-AIMD	✓			✓		PT
DistGraph		✓		✓		
ScaleMine		✓			✓	
T-FSM		✓			✓	
PrefixFPM			✓		✓	

Some recent systems such as G-thinker [53, 54], G-Miner [7] and Fractal [10] resolve this issue by allowing depth-first subgraph-instance backtracking without actually materializing the instances. While these systems target one-time offline analytics, G-thinkerQ [63] efficiently supports interactive online querying where users continually submit subgraph queries with different query contents. AutoMine [26], GraphPi [33] and GraphZero [25] focus on subgraph enumeration/matching where different vertex matching order leads to different costs, and they adopt a compilation-based approach to generate subgraph enumeration code with a favorable vertex matching order. Among them, G-thinker, G-Miner, and G-thinkerQ only supports SF but not FSM, while GraphPi and GraphZero are dedicated to subgraph enumeration/matching. There are also systems dedicated to FSM in a single big graph, such as ScaleMine [3], DistGraph [37] and T-FSM [65], with T-FSM being the most efficient which decomposes the problem of pattern support evaluation into subgraph-matching tasks for parallel computation by backtracking search, and which supports all pruning techniques of the FSM algorithm GraMi [11]. For FSM from a database of graph transactions (rather than a big graph), PrefixFPM [56, 57] provides an efficient parallel solution by depth-first pattern extension.

Recently, some systems begin to explore the use of GPUs to further accelerate subgraph enumeration/matching. Since backtracking was deemed not beneficial on GPUs [17], most current solutions such as GSI [67] and cuTS [45] maintain and grow the intermediately matched subgraphs in a BFS manner (in the subgraph extension search tree) to allow coalesced memory access. Since

Table 2: Techniques of Distributed GNN Training Systems

Systems	Graph Data Communication	Operators Scheduling	Model Computation and Synchronization	Other Optimizations	Full-Graph GNN
Euler		✓			
AliGraph		✓			
DistDGL	✓				
AGL	✓				
P^3	✓	✓	✓		
NeutronStar		✓			
ByteGNN	✓	✓			
DGCL	✓			✓	
BGL	✓	✓			
Sancus		✓	✓		
Dorylus		✓	✓	✓	
DistGNN	✓				✓
HongTu	✓				✓

GPU global memory has a limited space, PBE [15], VSGM [18] and SGSI [66] explore methods to partition a large input graph so that only a partition needs to be loaded to a GPU for processing at each time. G^2-AIMD [62] supports general SF by BFS-based subgraph extension, and it avoids intermediate subgraph-size explosion with novel system designs such as adaptive chunk-size adjustment and host-memory subgraph buffering.

More recently, DFS solutions are explored on GPUs such as STMatch [44], T-DFS [64] and EGSM [36], where each warp conducts DFS on a chunk of independent subtrees of the subgraph extension search tree (as tasks) by maintaining its own stack, and load balancing is achieved by work stealing which splits heavy tasks. While STMatch and T-DFS are pure DFS solutions, EGSM advocates a BFS-DFS hybrid solution where the more efficient BFS is used when device memory permits, and if memory becomes insufficient, it falls back to DFS to match the remaining query vertices.

Table 1 summarizes the key features of existing TLAG systems that we will cover in this tutorial.

3 SYSTEMS FOR GNN TRAINING

Graph classification and regression have been conventionally solved by shallow-learning models such as support vector machines [28, 29, 31]. Recent advancement in deep learning has made graph neural networks (GNNs) (e.g., GCN, GAT) popular as downstream models for graph machine learning. GNNs operate by collecting the features of neighboring vertices and connected edges, and recursively aggregating them and transforming them into new vertex features. Taking the GraphSAGE model [16] as an example, where each graph convolution layer can be expressed as follows:

$$\mathbf{h}_{\mathcal{N}(v)}^{(k)} \leftarrow \text{AGGREGATE}_k\big(\{\mathbf{h}_u^{(k-1)} \mid \forall u \subseteq \mathcal{N}(v)\}\big),$$
$$\mathbf{h}_v^{(k)} \leftarrow \sigma\Big(\mathbf{W}^{(k)} \cdot \text{CONCAT}\big(\mathbf{h}_v^{(k-1)}, \mathbf{h}_{\mathcal{N}(v)}^{(k)}\big)\Big),$$

where $\mathcal{N}(v)$ denotes the set of v's neighboring vertices, superscript (k) denotes the GNN layer number. We can see the each layer of GNN has two stages: *Graph Data Retrieving* (every vertex obtains the feature vectors of its neighbors from the previous layer for aggregation), and *Model Computation and Synchronization*.

A large number of GNN training systems have been proposed in recent years. However, most of them are single-GPU systems, which cannot scale to industrial-scale large graphs. In this tutorial, we will focus on distributed GNN training systems. We will first present the challenges and then introduce a range of representative GNN systems with a variety of techniques designed to address the key performance bottlenecks. These systems are from both academic and industrial contexts, providing a comprehensive coverage.

Distributed GNN training presents unique challenges different from traditional machine learning tasks. Unlike training tasks in computer vision or natural language, GNN training requires access to neighborhood information that is not independent across training samples. As a result, the first challenge in distributed GNN training is the need for efficient (1) **Graph Data Communication**. Another challenge is (2) **Operator Scheduling** which needs to balance tasks among computing nodes, including subgraph sampling, neighborhood feature aggregation, and model learning operations such as loss computation, gradient calculation and parameter updating, while making optimal use of available resources. The third task is (3) **Model Computation and Synchronization**, where frequent synchronizations between nodes during training are needed to ensure consistency in model parameters, which often results in increased synchronization delay and communication overhead.

To tackle the aforementioned major challenges, many techniques have been proposed in recent years, which we review next.

Graph Data Communications. Several techniques have been proposed to effectively manage the significant volumes of graph data communications. *Neighborhood sampling* is one of the most commonly employed techniques, as it limits the number of neighbors of each node used for training. This technique has been widely adopted in industrial GNN systems such as Alibaba's Euler [4] and AliGraph [73], and ByteGNN [71]. *Graph partitioning* has also been employed to enhance the efficiency of graph data communications. DistDGL [72] and DGCL [6] use traditional graph partitioning algorithms such as METIS [19] to minimize cross-machine data communication. However, since not every vertex in the graph is used for GNN training (GNN workload usually involves the neighborhood of training vertices within only a few hops), a global minimum edge-cut may not be the optimal choice for GNN training. To address this issue, ByteGNN [71] and BGL [22] propose heuristic algorithms to over-partition a graph into small blocks by performing BFS from train/validation/test seed vertices, till the BFS's meet (i.e., computing the graph Voronoi diagram of seed vertices), and then assign these blocks to workers in a streaming style. P^3's [13] method focuses primarily on dividing input data according to vertex feature rather than graph topology, aligning with its unique training technique that fuses model and data parallelism. Finally, AGL [68] employs the established infrastructure, MapReduce, to materialize the k-hop subgraphs for all training nodes before the actual training process, which eliminates the need for graph data communication during the training itself.

Operator Scheduling. Various systems utilize different execution scheduling policies and adopt the pipeline mechanism to making optimal use of available resources. Specifically, Euler [4], AliGraph [73] and ByteGNN [71] leverage operator abstraction to enable higher parallelism. ByteGNN further proposes a two-level scheduling scheme to control operator execution within and between iterations. NeutronStar [43] offers a flexible auto-differentiation

strategy by separating dependency management from graph operation and neural network functions. BGL [22], P^3 [13] and Dorylus [39] implement pipeline mechanisms but they differ in their underlying task units or computing units. BGL uses a factored paradigm where different tasks are processed on different types of executors. In contrast, task units in Dorylus are processed by several Serverless Threads. P^3 follows a similar model training pipeline as traditional deep learning training, but it splits the forward (and backward) process into two phases for model and data parallelism.

Model Computation. GNN models are relatively small compared to DNN models, and model computation only needs to be carried out on densely packed vectors. However, sampling a large graph involves accessing a large amount of data from random (including remote) locations to construct the neighborhood subgraph for each sampled seed vertex. Several systems have been developed on CPU clusters such as Euler [4], DistDGL [72], AliGraph [73] and ByteGNN [71]. The training well overlaps graph data retrieving and GNN computation by fine-grained task assignment. In contrast, DistGNN [27] targets full-graph training on CPU clusters, via an efficient shared memory implementation, communication reduction using a minimum vertex-cut graph partitioning algorithm and communication avoidance using a family of delayed-update algorithms. When models become larger, GPU computing power becomes indispensable and thus more recently GPU distributed GNN systems have been proposed. As an example, P^3 [13] proposes a unique approach for computation, referred to as the push-pull strategy, which combines the intra-layer model parallelism and data parallelism on GPUs. In contrast, HongTu [42] targets full-graph training on multiple GPUs, which stores vertex data in CPU memory and offloads training to GPUs.

Model Synchronization. To reduce the model synchronization overhead, several alternative asynchronous training paradigms have been developed. One of these paradigms is the bounded staleness approach, which is adopted in Dorylus [39] and P^3 [13]. Bounded staleness is an asynchronous communication paradigm that limits the use of outdated model weights in the training process. By doing so, it allows pipelining to be fully exploited while ensuring the convergence of the final model. To avoid static bounded staleness, Sancus [30] proposes a staleness-aware communication algorithm that dynamically adjusts the staleness based on variations in embeddings or gradients. This approach enables more efficient and effective asynchronous communication in the training process.

Other Techniques. Besides the aforementioned techniques, other optimizations can be employed to effectively train distributed GNN models. DGCL leverages high-bandwidth NVLINK hardware between GPUs to accelerate the speed of GNN training, through the use of special communication plans that are generated based on high link speed and topology. Dorylus [39] utilizes Lambda threads (serverless) service offered by cloud providers for computation. According to [39], the utilization of CPU servers and serverless function calls from clouds is a more cost-effective option than using GPUs. This feature allows for better scalability and cost-effectiveness for users, demonstrating that using cloud computing is an affordable way to achieve high-performance GNN training.

We will also introduce recent works for compressed GNN training using various quantization techniques, such as EC-Graph [34], EXACT [23], F^2CGT [24] and Sylvie [69].

4 TARGET AUDIENCE AND PREREQUISITES

The target audience for this tutorial includes anyone who are interested in large-scale graph analytics and machine learning, such as (1) researchers and practitioners who would like to understand how to choose the proper systems for their graph-related applications at hand, and (2) system developers who want to learn how to design graph-parallel systems. Our tutorial will be self-explanatory with minimal prerequisites, including a basic understanding of graph theory and some familiarity with GNNs.

5 PRIOR TUTORIALS AND DIFFERENCES

We have delivered this tutorial at IJCAI 2024, KDD 2024 [60], CIKM 2024 [59], and EDBT 2025 [59]. The audience is mainly from the data mining community currently, as well as the database community in Europe, so presenting at VLDB 2025 would expose this tutorial to a broader audience in the database community. This tutorial will cover more up-to-date systems not covered in previous tutorials.

6 FORMAT AND TUTORIAL LENGTH

Our tutorial will be in lecture-style format. We would like to request for two sessions (3 hours) so that we can cover each of the two system topics (structural analytics, and GNN) in one session. However, we are happy to reduce the scope of works covered to fit the two topics into one session (1.5 hours).

7 PRESENTERS AND THEIR EXPERTISE

This tutorial will be delivered by Dr. Da Yan from the Department of Computer Science at Indiana University Bloomington, and his graph systems team members Lyuheng Yuan, Akhlaque Ahmad and Saugat Adhikari. Dr. Yan and his team have developed graph-analytics systems including G-thinker [14, 20, 53, 54], G-thinkerQ [63], PrefixFPM [56, 57], T-FSM [21, 65], G^2-AIMD [62] and T-DFS [64], and are the pioneers of the think-like-a-task computing model T-thinker. Dr. Yan also has extensive experience in developing think-like-a-vertex (TLAV) graph systems and GNN. Dr. Yan and his team have a long history of developing graph-parallel systems, with dozens of related publications in top conferences such as SIGMOD, VLDB, KDD, ICDE, and top journals such as ACM TODS, VLDB Journal, IEEE TPDS and IEEE TKDE. Dr. Yan led the development of the BigGraph@CUHK platform [1] with many well-known systems following the TLAV model such as Blogel [49], Pregel+ [50], Quegel [51, 70], GraphD [55] and LWCP [48]. He has also delivered a tutorial on TLAV systems in SIGMOD 2016 [47], and published books on this topic with prestigious publishers [46, 58]. Dr. Yan is the sole winner of Hong Kong 2015 Young Scientist Award in Physical/Mathematical science, and his graph systems research was funded by the DOE Early Career Research Program in 2023.

ACKNOWLEDGMENTS

This work was supported by DOE ECRP Award DE-SC0025228, NSF OAC-2414474, NSF OAC-2414185 and 2024–2025 Luddy Faculty Fellow Award from Indiana University Bloomington.

REFERENCES

[1] [n.d.]. BigGraph@CUHK. http://www.cse.cuhk.edu.hk/systems/graph/.

[2] [n.d.]. GNN Tutorials. https://graph-neural-networks.github.io/.

[3] Ehab Abdelhamid, Ibrahim Abdelaziz, Panos Kalnis, Zuhair Khayyat, and Fuad T. Jamour. 2016. Scalemine: scalable parallel frequent subgraph mining in a single large graph. In *SC*. 716–727.

[4] Alibaba. 2020. Euler. https://github.com/alibaba/euler.

[5] Emily Alsentzer, Samuel G. Finlayson, Michelle M. Li, and Marinka Zitnik. 2020. Subgraph Neural Networks. In *NeurIPS*.

[6] 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 '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021*. ACM, 130–144. https://doi.org/10.1145/3447786.3456233

[7] Hongzhi Chen, Miao Liu, Yunjian Zhao, Xiao Yan, Da Yan, and James Cheng. 2018. G-Miner: an efficient task-oriented graph mining system. In *EuroSys*. ACM, 32:1–32:12.

[8] Xuhao Chen, Roshan Dathathri, Gurbinder Gill, and Keshav Pingali. 2020. Pangolin: An Efficient and Flexible Graph Mining System on CPU and GPU. *Proc. VLDB Endow.* 13, 8 (2020), 1190–1205.

[9] Shumo Chu and James Cheng. 2012. Triangle listing in massive networks. *ACM Trans. Knowl. Discov. Data* 6, 4 (2012), 17:1–17:32.

[10] Vinícius Vitor dos Santos Dias, Carlos H. C. Teixeira, Dorgival O. Guedes, Wagner Meira Jr., and Srinivasan Parthasarathy. 2019. Fractal: A General-Purpose Graph Pattern Mining System. In *SIGMOD Conference 2019*. ACM, 1357–1374.

[11] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. 2014. GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph. *Proc. VLDB Endow.* 7, 7 (2014), 517–528.

[12] Fabrizio Frasca, Beatrice Bevilacqua, Michael M. Bronstein, and Haggai Maron. 2022. Understanding and Extending Subgraph GNNs by Rethinking Their Symmetries. In *NeurIPS*.

[13] Swapnil Gandhi and Anand Padmanabha Iyer. 2021. P3: Distributed Deep Graph Learning at Scale. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*. USENIX Association, 551–568. https://www.usenix.org/conference/osdi21/presentation/gandhi

[14] Guimu Guo, Da Yan, M. Tamer Özsu, Zhe Jiang, and Jalal Khalil. 2020. Scalable Mining of Maximal Quasi-Cliques: An Algorithm-System Codesign Approach. *Proc. VLDB Endow.* 14, 4 (2020), 573–585.

[15] Wentian Guo, Yuchen Li, Mo Sha, Bingsheng He, Xiaokui Xiao, and Kian-Lee Tan. 2020. GPU-Accelerated Subgraph Enumeration on Partitioned Graphs. In *SIGMOD*. ACM, 1067–1082.

[16] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 1024–1034. https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html

[17] John Jenkins, Isha Arkatkar, John D. Owens, Alok N. Choudhary, and Nagiza F. Samatova. 2011. Lessons Learned from Exploring the Backtracking Paradigm on the GPU. In *Euro-Par 2011 Parallel Processing - 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011, Proceedings, Part II (Lecture Notes in Computer Science)*, Vol. 6853. Springer, 425–437.

[18] Guanxian Jiang, China Qihui Zhou, Tatiana Jin, Boyang Li, Yunjian Zhao, Yichao Li, and James Cheng. 2022. VSGM: View-Based GPU-Accelerated Subgraph Matching on Large Graphs. In *SC*. IEEE Computer Society, 739–753.

[19] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392. https://doi.org/10.1137/S1064827595287997

[20] Jalal Khalil, Da Yan, Guimu Guo, and Lyuheng Yuan. 2022. Parallel mining of large maximal quasi-cliques. *VLDB J.* 31, 4 (2022), 649–674.

[21] Jalal Khalil, Da Yan, Lyuheng Yuan, Jiao Han, Saugat Adhikari, Cheng Long, and Yang Zhou. 2024. FSM-Explorer: An Interactive Tool for Frequent Subgraph Pattern Mining From a Big Graph. In *ICDE*. IEEE, 5405–5408.

[22] Tianfeng Liu, Yangrui Chen, Dan Li, Chuan Wu, Yibo Zhu, Jun He, Yanghua Peng, Hongzheng Chen, Hongzhi Chen, and Chuanxiong Guo. 2021. BGL: GPU-Efficient GNN Training by Optimizing Graph Data I/O and Preprocessing. *CoRR* abs/2112.08541 (2021). arXiv:2112.08541 https://arxiv.org/abs/2112.08541

[23] Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. 2022. EXACT: Scalable Graph Neural Networks Training via Extreme Activation Compression. In *ICLR*. OpenReview.net.

[24] Yuxin Ma, Ping Gong, Tianming Wu, Jiawei Yi, Chengru Yang, Cheng Li, Qirong Peng, Guiming Xie, Yongcheng Bao, Haifeng Liu, and Yinlong Xu. 2024. Eliminating Data Processing Bottlenecks in GNN Training over Large Graphs via Two-level Feature Compression. *Proc. VLDB Endow.* 17, 11 (2024), 2854–2866.

[25] Daniel Mawhirter, Sam Reinehr, Connor Holmes, Tongping Liu, and Bo Wu. 2021. GraphZero: A High-Performance Subgraph Matching System. *ACM SIGOPS Oper. Syst. Rev.* 55, 1 (2021), 21–37.

[26] Daniel Mawhirter and Bo Wu. 2019. AutoMine: harmonizing high-level abstraction and high performance for graph mining. In *SOSP*. ACM, 509–523.

[27] Vasimuddin Md, Sanchit Misra, Guixiang Ma, Ramanarayan Mohanty, Evangelos Georganas, Alexander Heinecke, Dhiraj D. Kalamkar, Nesreen K. Ahmed, and Sasikanth Avancha. 2021. DistGNN: scalable distributed training for large-scale graph neural networks. In *SC*. ACM, 76.

[28] Shirui Pan and Xingquan Zhu. 2013. Graph Classification with Imbalanced Class Distributions and Noise. In *IJCAI*, Francesca Rossi (Ed.). IJCAI/AAAI, 1586–1592.

[29] Martin S. R. Paradesi, Doina Caragea, and William H. Hsu. 2007. Structural Prediction of Protein-Protein Interactions in Saccharomyces cerevisiae. In *IEEE BIBE*. IEEE Computer Society, 1270–1274.

[30] 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. *Proc. VLDB Endow.* 15, 9 (2022), 1937–1950. https://www.vldb.org/pvldb/vol15/p1937-peng.pdf

[31] Hiroto Saigo, Sebastian Nowozin, Tadashi Kadowaki, Taku Kudo, and Koji Tsuda. 2009. gBoost: a mathematical programming approach to graph classification and regression. *Mach. Learn.* 75, 1 (2009), 69–89.

[32] Yanyan Shen, Lei Chen, Jingzhi Fang, Xin Zhang, Shihong Gao, and Hongbo Yin. 2024. Efficient Training of Graph Neural Networks on Large Graphs. *Proc. VLDB Endow.* 17, 12 (2024), 4237–4240.

[33] Tianhui Shi, Mingshu Zhai, Yi Xu, and Jidong Zhai. 2020. GraphPi: high performance graph pattern matching through effective redundancy elimination. In *SC*. IEEE/ACM, 100.

[34] Zhen Song, Yu Gu, Jianzhong Qi, Zhigang Wang, and Ge Yu. 2022. EC-Graph: A Distributed Graph Neural Network System with Error-Compensated Compression. In *ICDE*. IEEE, 648–660.

[35] Andrew Stolman, Caleb Levy, C. Seshadhri, and Aneesh Sharma. 2022. Classic Graph Structural Features Outperform Factorization-Based Graph Embedding Methods on Community Labeling. In *SDM*. SIAM, 388–396.

[36] Xibo Sun and Qiong Luo. 2023. Efficient GPU-Accelerated Subgraph Matching. *Proc. ACM Manag. Data* 1, 2 (2023), 181:1–181:26.

[37] Nilothpal Talukder and Mohammed J. Zaki. 2016. A distributed approach for graph mining in massive networks. *Data Min. Knowl. Discov.* 30, 5 (2016), 1024–1052.

[38] Carlos H. C. Teixeira, Alexandre J. Fonseca, Marco Serafini, Georgos Siganos, Mohammed J. Zaki, and Ashraf Aboulnaga. 2015. Arabesque: a system for distributed graph mining. In *SOSP*. ACM, 425–440.

[39] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*. USENIX Association, 495–514. https://www.usenix.org/conference/osdi21/presentation/thorpe

[40] Hanchen Wang, Rong Hu, Ying Zhang, Lu Qin, Wei Wang, and Wenjie Zhang. 2022. Neural Subgraph Counting with Wasserstein Estimator. In *SIGMOD*. ACM, 160–175.

[41] Kai Wang, Zhiqiang Zuo, John Thorpe, Tien Quang Nguyen, and Guoqing Harry Xu. 2018. RStream: Marrying Relational Algebra with Streaming for Efficient Graph Mining on a Single Machine. In *OSDI*. USENIX Association, 763–782.

[42] Qiange Wang, Yao Chen, Weng-Fai Wong, and Bingsheng He. 2023. HongTu: Scalable Full-Graph GNN Training on Multiple GPUs. *Proc. ACM Manag. Data* 1, 4 (2023), 246:1–246:27.

[43] 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*. ACM, 1301–1315. https://doi.org/10.1145/3514221.3526134

[44] Yihua Wei and Peng Jiang. 2022. STMatch: Accelerating Graph Pattern Matching on GPU with Stack-Based Loop Optimizations. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, November 13-18, 2022*, Felix Wolf, Sameer Shende, Candace Culhane, Sadaf R. Alam, and Heike Jagode (Eds.). IEEE, 53:1–53:13.

[45] Lizhi Xiang, Arif Khan, Edoardo Serra, Mahantesh Halappanavar, and Aravind Sukumaran-Rajam. 2021. cuTS: scaling subgraph isomorphism on distributed multi-GPU systems using trie based data structure. In *SC*. ACM, 69.

[46] Da Yan, Yingyi Bu, Yuanyuan Tian, and Amol Deshpande. 2017. Big Graph Analytics Platforms. *Found. Trends Databases* 7, 1-2 (2017), 1–195.

[47] Da Yan, Yingyi Bu, Yuanyuan Tian, Amol Deshpande, and James Cheng. 2016. Big Graph Analytics Systems. In *SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 2241–2243.

[48] Da Yan, James Cheng, Hongzhi Chen, Cheng Long, and Purushotham V. Bangalore. 2019. Lightweight Fault Tolerance in Pregel-Like Systems. In *ICPP*. ACM, 69:1–69:10.

[49] Da Yan, James Cheng, Yi Lu, and Wilfred Ng. 2014. Blogel: A Block-Centric Framework for Distributed Computation on Real-World Graphs. *Proc. VLDB Endow.* 7, 14 (2014), 1981–1992.

[50] Da Yan, James Cheng, Yi Lu, and Wilfred Ng. 2015. Effective Techniques for Message Reduction and Load Balancing in Distributed Graph Computation. In *WWW*, Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi (Eds.). ACM, 1307–1317.

[51] Da Yan, James Cheng, M. Tamer Özsu, Fan Yang, Yi Lu, John C. S. Lui, Qizhen Zhang, and Wilfred Ng. 2016. A General-Purpose Query-Centric Framework for Querying Big Graphs. *Proc. VLDB Endow.* 9, 7 (2016), 564–575.

[52] Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. 2014. Pregel Algorithms for Graph Connectivity Problems with Performance Guarantees. *Proc. VLDB Endow.* 7, 14 (2014), 1821–1832.

[53] Da Yan, Guimu Guo, Md Mashiur Rahman Chowdhury, M. Tamer Özsu, Wei-Shinn Ku, and John C. S. Lui. 2020. G-thinker: A Distributed Framework for Mining Subgraphs in a Big Graph. In *ICDE 2020*. IEEE, 1369–1380.

[54] Da Yan, Guimu Guo, Jalal Khalil, M. Tamer Özsu, Wei-Shinn Ku, and John C. S. Lui. 2022. G-thinker: a general distributed framework for finding qualified subgraphs in a big graph with load balancing. *VLDB J.* 31, 2 (2022), 287–320.

[55] Da Yan, Yuzhen Huang, Miao Liu, Hongzhi Chen, James Cheng, Huanhuan Wu, and Chengcui Zhang. 2018. GraphD: Distributed Vertex-Centric Graph Processing Beyond the Memory Limit. *IEEE Trans. Parallel Distributed Syst.* 29, 1 (2018), 99–114.

[56] Da Yan, Wenwen Qu, Guimu Guo, and Xiaoling Wang. 2020. PrefixFPM: A Parallel Framework for General-Purpose Frequent Pattern Mining. In *ICDE 2020*. IEEE, 1938–1941.

[57] Da Yan, Wenwen Qu, Guimu Guo, Xiaoling Wang, and Yang Zhou. 2022. PrefixFPM: a parallel framework for general-purpose mining of frequent and closed patterns. *VLDB J.* 31, 2 (2022), 253–286.

[58] Da Yan, Yuanyuan Tian, and James Cheng. 2017. *Systems for Big Graph Analytics*. Springer. https://doi.org/10.1007/978-3-319-58217-7

[59] Da Yan, Lyuheng Yuan, Akhlaque Ahmad, and Saugat Adhikari. 2024. Systems for Scalable Graph Analytics and Machine Learning: Trends and Methods. In *CIKM*. ACM, 5547–5550.

[60] Da Yan, Lyuheng Yuan, Akhlaque Ahmad, Chenguang Zheng, Hongzhi Chen, and James Cheng. 2024. Systems for Scalable Graph Analytics and Machine Learning: Trends and Methods. In *KDD*. ACM, 6627–6632.

[61] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. 2020. Neural Subgraph Matching. *CoRR* abs/2007.03092 (2020). arXiv:2007.03092 https://arxiv.org/abs/2007.03092

[62] Lyuheng Yuan, Akhlaque Ahmad, Da Yan, Jiao Han, Saugat Adhikari, Xiaodong Yu, and Yang Zhou. 2024. G^2-AIMD: A Memory-Efficient Subgraph-Centric Framework for Efficient Subgraph Finding on GPUs. In *ICDE*. IEEE, 3164–3177.

[63] Lyuheng Yuan, Guimu Guo, Da Yan, Saugat Adhikari, Jalal Khalil, Cheng Long, and Lei Zou. 2025. G-thinkerQ: A General Subgraph Querying System with a Unified Task-Based Programming Model. *IEEE Trans. Knowl. Data Eng., accepted and to appear* (2025).

[64] Lyuheng Yuan, Da Yan, Jiao Han, Akhlaque Ahmad, Yang Zhou, and Zhe Jiang. 2024. Faster Depth-First Subgraph Matching on GPUs. In *ICDE*. IEEE, 3151–3163.

[65] Lyuheng Yuan, Da Yan, Wenwen Qu, Saugat Adhikari, Jalal Khalil, Cheng Long, and Xiaoling Wang. 2023. T-FSM: A Task-Based System for Massively Parallel Frequent Subgraph Pattern Mining from a Big Graph. *Proc. ACM Manag. Data* 1, 1, 74:1–74:26.

[66] Li Zeng, Lei Zou, and M Tamer Özsu. 2022. SGSI–A Scalable GPU-friendly Subgraph Isomorphism Algorithm. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[67] Li Zeng, Lei Zou, M. Tamer Özsu, Lin Hu, and Fan Zhang. 2020. GSI: GPU-friendly Subgraph Isomorphism. In *ICDE*. IEEE, 1249–1260.

[68] Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. 2020. AGL: A Scalable System for Industrial-purpose Graph Machine Learning. *Proc. VLDB Endow.* 13, 12 (2020), 3125–3137. https://doi.org/10.14778/3415478.3415539

[69] Meng Zhang, Qinghao Hu, Cheng Wan, Haozhao Wang, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2024. Sylvie: 3D-Adaptive and Universal System for Large-Scale Graph Neural Network Training. In *ICDE*. IEEE, 3823–3836.

[70] Qizhen Zhang, Da Yan, and James Cheng. 2016. Quegel: A General-Purpose System for Querying Big Graphs. In *SIGMOD*. ACM, 2189–2192.

[71] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. 2022. ByteGNN: Efficient Graph Neural Network Training at Large Scale. *Proc. VLDB Endow.* 15, 6 (2022), 1228–1242. https://www.vldb.org/pvldb/vol15/p1228-zheng.pdf

[72] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs. *CoRR* abs/2010.05337 (2020). arXiv:2010.05337 https://arxiv.org/abs/2010.05337

[73] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: A Comprehensive Graph Neural Network Platform. *Proc. VLDB Endow.* 12, 12 (2019), 2094–2105. https://doi.org/10.14778/3352063.3352127