Technical Report

October 30, 2024

1 Evaluation

1.1 Experimental Setting

Experimental Platform. Table 1 illustrates the evaluation platforms including two single machines and one eight-machine cluster.

GNN Models. We use three sampling-based GNN models: GAT [21], GraphSAGE [9], and GCN [13]. All models adopt a 2-hop random neighbor sampling by default. The sampling fan-outs are 25 and 10. The hidden dimension of GAT is set to 64 and the head number of each layer is set to 8. The hidden dimensions of GraphSAGE and GCN are set to 256. Similar to existing work [24, 19], the batch size is set to 8000 by default. Node classification is used as the GNN task.

Datasets. We conduct our experiments on multiple real-world graph datasets with various scales. Table 2 shows the dataset characteristics. The Paper100M (PA) is available in Open Graph Benchmark [10]. The IGB-HOM (IG) is from the IGB dataset [12]. The UK-2014 (UK), and Clue-web (CL) are from WebGraph [7, 6, 4, 5]. The LDBC-SNB-Bi-SF3000 (LD) is available at the LDBC social network benchmarks [3]. Because UK, CL, and LD have no feature, we manually generate the features with the dimension specified as 1024, following IG's setting. Similar to [24]'s setting, we randomly choose 1% of vertices from each graph as training vertices. We also adapt the feature dimension of PA, IG, and UK for experiments in 1.2 and 1.3. We mark these variants as Name-Dimension. For instance, PA-1024 refers to a variant of PA whose feature dimension is 1024.

Baselines. We use the state-of-the-art out-of-core GNN systems, Ginex [15], MariusGNN [22] and GIDS [14] and distributed system DistDGL [25] as the baseline systems.

1.2 Comparison of End-to-end Throughput

Comparing to Out-of-core Baselines. We compare the throughput of Hyperion with all out-of-core three baselines [15, 22, 14] in machine A. We report the average training epoch time of all compared systems on all datasets in Table 2 and three GNN models, as illustrated by Figure 1. We use 12 P5510 SSDs [18] to store the datasets. We set the number of CPU threads to 96 for CPU-managed baselines Ginex and MariusGNN.

We first examine the average epoch time of each system on terabyte-scale datasets (IG, UK, CL, LD). We observe that Hyperion outperforms GIDS by up to 3.1×. Moreover, GIDS runs out of GPU memory on larger UK, CL, and LD datasets, because BaM [17] requires over 80GB metadata in the page cache design. MariusGNN runs out of memory due to large memory consumption during pre-processing on these datasets. Compared to CPU-managed baseline Ginex, Hyperion achieves over 167× speedup.

On the smallest dataset PA, all systems can store all topology and features in CPU memory. In this case, Hyperion outperforms GIDS, MariusGNN, and Ginex by up to 2.3×, 15.9×, and 68×, respectively, indicating that Hyperion still has superior throughput on small graphs.

Comparing to In-memory System. We compare Hyperion with SOTA in-memory system Legion [19] to show that Hyperion can achieve in-memory-like training throughput by adding cheap NVMe SSDs. We evaluate in machine A with 12× P5510 SSDs. We report two GNN models due to space limitations. We evaluate four large-scale datasets in Table 2 of up to 700 GB by adapting the feature dimension under the constraint of CPU memory (768 GB). For example, PA-1024 represents adapts the feature dimension of original PA datasets to 1024. For both systems, we fill all the available GPU memory as the cache. Hyperion only utilizes 128 GB CPU memory as the cache while Legion stores all the feature/topology data in CPU memory.

Figure 2 illustrates the average epoch time of Hyperion compared to Legion. We observe that Hyperion can achieve 88%~97% throughput of Legion with both GNN models, indicating that out-of-core training with Hyperion can achieve close training throughput to in-memory systems.

1.3 Comparison of End-to-end TPC

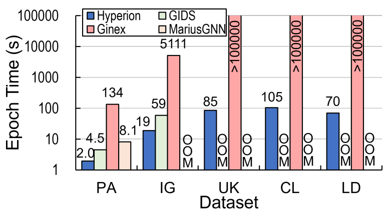
We compare the end-to-end throughput and TPCof Hyperion with out-of-core and distributed baselines[1]. We calculate the TPCbased on five-year TCO (See § 2 and Table 1). Hyperion and GIDS run on machine A with 12 P5510 SSDs and DistDGL runs on cluster C using 8 machines. As DistDGL requires CPU to execute distributed sampling, we maximize the CPU thread number per machine to 48. We measure the network utilization of DistDGL by Intel PCM [11] and find that DistDGL only reaches 20Gbps peak network throughput so DistDGL would not be bottlenecked by the PCIe 3.0 bandwidth of cluster C. Because DistDGL runs out of memory on IG due to memory overhead (we find DistDGL allocates about 5× memory compared to the original dataset size), we also evaluate IG-256 whose feature dimension is 256.

Figure 3 demonstrates that Hyperion outperforms the TPCof GIDS by up to 7.7× and outperforms DistDGL by up to 60×, indicating that Hyperion can maximize TPCfor GNN training. Hyperion achieves higher throughput than DistDGL because DistDGL's CPU-based distributed graph sampling is less efficient than GPU-based sampling in Hyperion [24, 16, 8]. Though DistDGL can achieve higher throughput than Hyperion with more machines, the TPCof DistDGL can hardly increase because the monetary cost grows linearly

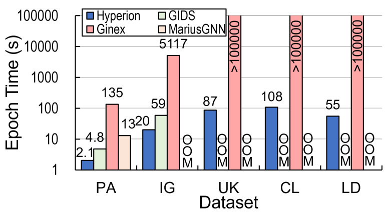
[1]For out-of-core baselines, we only the report GIDS because Figure 1 proves Ginex and MariusGNN are significantly slower than GIDS and Hyperion in the same machine

Table 1: Detailed evaluation platforms.

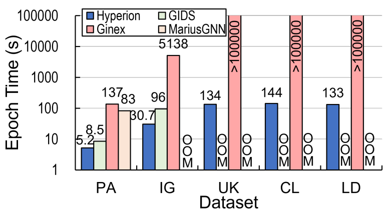
Machine/Cluster	**A**	**B**	**C (Cluster, Eight Machines)**
GPU	80GB-PCIe-A100	80GB-PCIe-H800	80GB-PCIe-A100
SSD	12 × 3.84TB Intel P5510, 6 × 1TB Samsung 980pro	12 × 3.84TB Intel P5510	/
PCIe/Network	4.0x16	5.0x16	3.0x16, 100Gbps
CPU	Intel(R) Xeon(R) Gold 5320 CPU (2 × 52 threads) @ 2.20GHz	Intel(R) Xeon(R) Gold 6426Y CPU (2 × 32 threads) @ 2.50GHZ	Intel(R) Xeon(R) Silver 4214 CPU (2 × 24 threads) @ 2.20GHz
CPU Mem.	768GB	512GB	256GB
TCO	48,971 (with 12 × Intel P5510)	78,794 (with 12 × Intel P5510)	45,275 × 8



(a) GraphSAGE



(b) GCN

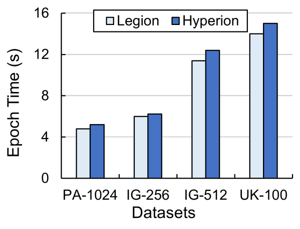


(c) GAT

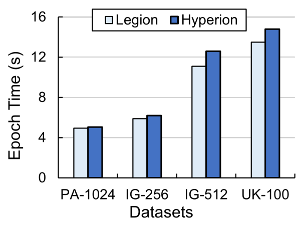
Figure 1: End-to-end Throughput of Hyperion, GIDS, MariusGNN, and Ginex.

Table 2: Detailed information of datasets

Dataset	PA	IG	UK	CL	LD
Vertices Num.	111M	269M	0.79B	1B	5.6B
Edges Num.	1.6B	4B	47.2B	42.5B	10B
Topo. Size	14GB	34GB	384GB	348GB	125GB
Feat. Dim.	128	1024	1024	1024	1024
Feat. Size	56GB	1.1TB	3.2TB	4.1TB	23TB



(a) GraphSAGE



(b) GCN

Figure 2: Epoch Time: Hyperion vs. In-memory System Legion.

with machine number. We evaluated the PA dataset using the three GNN models and observed that the throughput of DistDGL scales nearly linearly with 1 to 8 machines, reaching 87% of Hyperion's throughput when using 8 machines. Though DistDGL would achieve higher throughput than Hyperion with over 8 machines, Hyperion can consistently outperform the TPCof DistDGL by over 18×.

1.4 Impact of Hyperion's Disk IO Stack

In this experiment, we evaluate the effect of GPU-initiated pipeline-friendly asynchronous disk IO stack. There are three evaluation metrics: 1) the impact of our IO stack on the GNN pipeline, 2) the achieved disk IO throughput, and 3) the used ratio of GPU computation resources of our design.

1.4.1 Impact on the GNN Pipeline

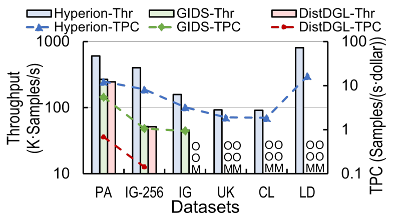
To examine the impact of Hyperion's pipeline-friendly asynchronous IO stack on the GNN pipeline, our comparison introduces Hyperion-NoPipe that launches all GPU cores for all stages in the GNN training process and executes all GPU-initiated operators in a serial order. We conduct the experiments on machine A with 12 P5510 SSDs.

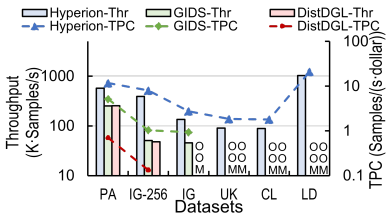
Figure 4 illustrates the comparison results under three GNN models and all datasets. We observe that Hyperion outperforms Hyperion-NoPipe by up to 1.74× on GraphSAGE, up to 1.39× on GCN, and up to 1.23× on GAT, because Hyperion can overlap GNN computation and disk IO accesses.

1.4.2 Achieved IO Throughput and GPU Utilization

Experimental Setting. We examine the achieved disk IO throughput and the used ratio of GPU computation resources on machine A with two different kinds of SSDs, i.e., Intel P5510 and Samsung 980pro. GIDS exhausts all GPU cores for IO kernels (256K thread blocks, 100% GPU core utilization). For a fair comparison, Hyperion starts the IO completion kernel immediately after the IO submission kernel and records the IO throughput for the overall process. For the IO submission kernel, we set the GPU thread block number to 1 and 128, which is about 1% and 100% GPU core utilization. For the IO completion kernel, we set the thread block number to 32 and 128, which is about 30% and 100% GPU core utilization, respectively. On P5510 (or 980pro), we fix the feature dimension to 1024 and vary the SSD numbers from 1 to 12 (or 6), as shown in Figure 5(a) (or Figure 5(c)). Besides, we fix the SSD number to 12 (6) and vary the feature dimension from 128 to 1024, as illustrated by Figure 5(b) (or Figure 5(d)).

Overall IO Throughput. Figure 5 shows that Hyperion's IO stack outperforms GIDS's IO stack under different numbers of SSDs and feature dimensions, because the Hyperion's IO


(a) GraphSAGE


(b) GCN

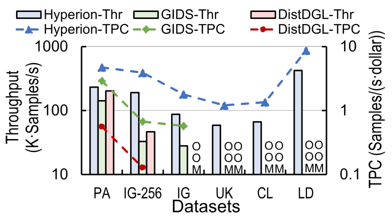
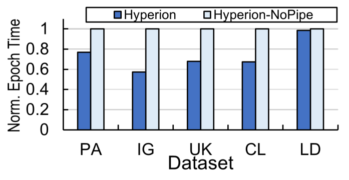
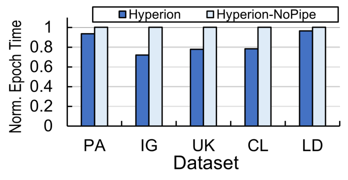

(c) GAT

Figure 3: End-to-end Throughput and TPCof Hyperion, GIDS, and DistDGL. X-Thr and X-TPC present the system throughput and TPC, respectively.


(a) GraphSAGE


(b) GCN

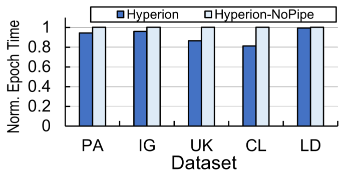
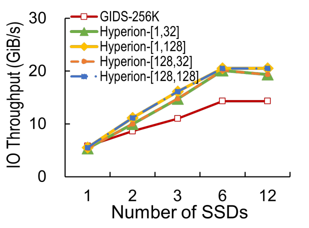
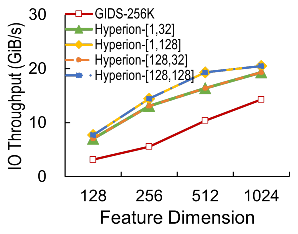
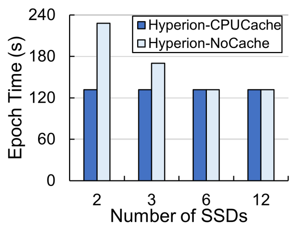
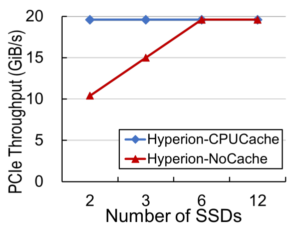

(c) GAT

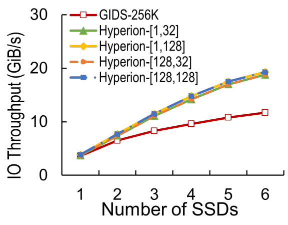
Figure 4: Impact of Deep GNN-aware Pipeline.

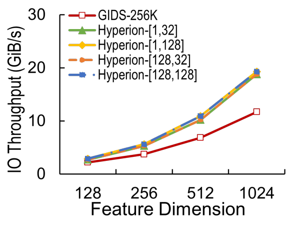

(a) IO Throughput w.r.t SSD Number, Feature Dimension of 1024, P5510

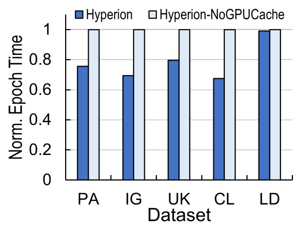

(b) IO Throughput w.r.t Feature Dimension, 12 SSDs, P5510


(a) Impact of CPU Cache on Epoch Time


(b) Impact of CPU Cache on GPU PCIe Throughput


(c) IO Throughput w.r.t SSD Number, Dimension = 1024, 980pro


(d) IO Throughput w.r.t Feature Dimension, 6 SSDs, 980pro


(c) Impact of GPU Cache.

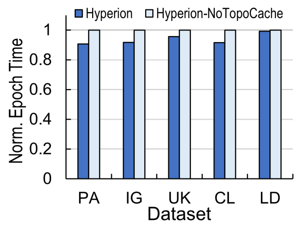

(d) Impact of Topology Cache.

Figure 6: Impact of Unified Cache.

Figure 5: Comparison of Overall IO Throughput and GPU Core Utilization of Hyperion and GIDS. Hyperion-[x,y] represents utilizing x and y thread blocks for the IO submission and completion kernels, respectively. GIDS-256K means that GIDS utilizes 256K thread blocks for the IO kernel.

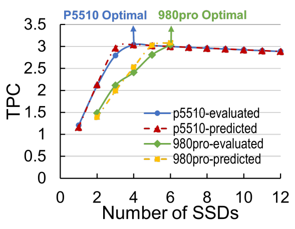
stack design can submit sufficient parallel IO requests to maximize the disk IO throughput. As shown in Figure 5(a), the IO throughput of Hyperion from 6 to 12 SSDs is almost the same due to the saturation of PCIe 4.0.

GPU Core Utilization. Figure 5 also proves that Hyperion only needs 1 thread block (1% GPU cores) for IO submission kernels to achieve a comparable IO throughput with 128 thread blocks (all GPU cores). Meanwhile, the IO completion kernel only needs 32 thread blocks (30% GPU cores) to reach an almost maximal throughput. The slight overhead is the data movement from the IO stack's internal buffer to the output feature buffer. In conclusion, the IO stack design enables Hyperion to maximize the disk IO throughput while leaving the majority of GPU cores for other useful GNN kernels, rather than waiting for the completion of IO commands.

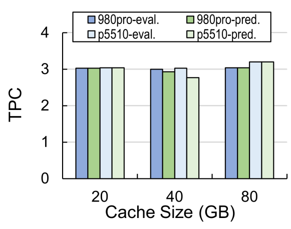
1.5 Impact of Hyperion's Cache

Hyperion proposes a unified cache that takes topology/feature and CPU/GPU memory into account. We evaluate the impact of components in the unified cache: CPU cache, GPU cache, and topology cache.

Impact of CPU Cache. To illustrate the impact of CPU cache, we propose two implementations: Hyperion-CPUCache and Hyperion-NoCache. Hyperion-CPUCache reads features from both the CPU cache and SSDs, while Hyperion-NoCache reads features only from SSDs. We disable GPU caches and maintain CPU cache for all graph topology data in both implementations. We use all the available CPU memory for the CPU feature. Figures 6 show the evaluation results on a terabyte-scale CL dataset (4.1TB). We vary the number of P5510 SSDs from 2 to 12. Figure 6(a)

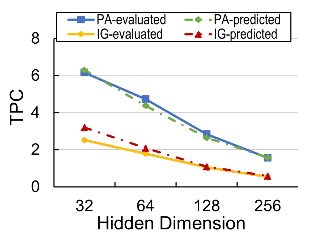


(a) TPC w.r.t. SSD number

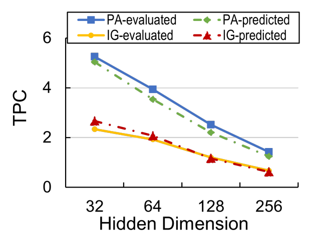


(b) TPC w.r.t. Cache Size

Figure 7: Case Study of IO bound.



(a) Machine A, PCIe 4.0, A100



(b) Machine B, PCIe 5.0, H800

Figure 8: Case Study of Computation Bound. TPCof GAT model with Various Model Sizes on Machines A and B.

illustrates that Hyperion-CPUCache outperforms Hyperion-NoCache by up to 1.73× and maintains a stable throughput even with only two SSDs. The underlying reason is that Hyperion-CPUCache reaches the maximal PCIe throughput under different numbers of SSDs, as shown in Figure 6(b).

Impact of GPU Cache. We examine the impact of GPU cache on different datasets with 12× P5510 SSDs. The GPU cache sizes are set to all the GPU memory except other GPU buffers like the GNN model and mini batch buffers while the CPU cache is set to all the available CPU memory. Figure 6(c) illustrates that Hyperion with GPU cache (Hyperion) can achieve speedup up to 1.48×, compared to Hyperion without GPU cache.

Impact of Topology Cache. We examine the impact of the topology cache. We use 12× P5510 SSDs. We compare Hyperion to Hyperion-NoTopoCache. For both systems, we use all the available GPU/CPU memory for cache. For Hyperion-NoTopoCache, we fill the CPU cache with feature data and access topology data from SSDs. Figure 6(d) illustrates that Hyperion outperforms Hyperion-NoTopoCache by up to 1.1× because accessing topology data from SSDs incurs more IO amplification compared to features.

1.6 Validation of TPC-analytical Model

We validate the predicting accuracy of the TPC-analytical model on various hardware combinations and different GNN models. We comprehensively examine the TPC-prediction under IO bound cases and computation bound cases.

Case study: IO Bound. We examine the TPCof GraphSAGE training on machine A with 1 to 12 P5510 SSDs and 2 to 6 980pro SSDs. In this series of experiments, GraphSAGE model training can be overlapped with PCIe IO, representing IO-bound cases of GNN training. First, we fix the GPU/CPU cache sizes to 40GB/20GB and report the predicted/evaluated TPCwith different numbers of SSDs. Figure 7(a) demonstrates that Hyperion can precisely predict the TPCwith different SSD numbers and can find the optimal numbers of SSDs that maximize the TPC. Second, we fix the GPU cache size and vary the CPU cache from 20 to 80 GB. Under each cache setting, Hyperion will predict an optimal number of SSDs that maximizes TPCand we also manually evaluate the optimal number of SSDs. Then we compare the predicted optimal TPCunder Hyperion's suggestion to the manually evaluated optimal TPCin each cache setting. Figure 7(b) shows that Hyperion can select a suitable number of SSDs that achieves over 90% TPCto the evaluated ones.

- **Findings 1**: Adding cheaper consumer-grade SSDs 980 pro can achieve similar TPCas enterprise-grade SSDs P5510.

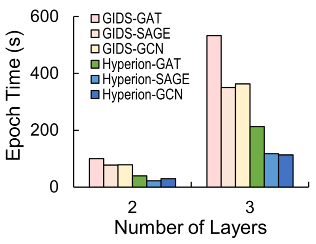


Figure 9: Impact of Model Layers on IG [12].

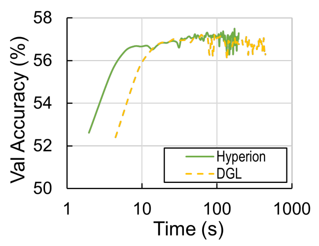


Figure 10: Impact of Model Convergence on PA [10].

Case Study: Computation Bound. We examine the TPCof GAT training on machines A and B. We use 12× P5510 SSDs in both machines. We vary the model sizes of GAT, i.e., hidden dimensions, from 32 to 256. From hidden dimensions 64 to 256, the GNN training is bounded by the model computation. We report two datasets: PA and IG. Figure 8 shows that Hyperion can precisely predict the TPCin different machines with different GPUs and PCIe throughput.

- **Findings 2**: Utilizing advanced H800 GPU can even achieve lower TPCcompared to A100 GPU due to higher costs.

1.7 Additional Experiments

We include experiments regarding the impact of model layers, model convergence, and pre-processing cost in this subsection.

Impact of GNN Layer Numbers. We compare Hyperion to GIDS with more layers on machine A with 12 P5510 SSDs and use the dataset IG. We set the batch size to 1000 because both systems with batch size 8000 run out of GPU memory on three-hop GAT. We set the two-hop/three-hop sampling fanout to (25,10) and (25,10,5). Figure 9 shows that Hyperion outperforms GIDS by up to 3.5× with two layers and up to 3.2× with three layers.

Model Convergence. In this experiment, we examine the model convergence of Hyperion on a popular benchmark PA [10]. Figure 10 shows the validation accuracy w.r.t. training time. Hyperion can converge to the same validation accuracy compared to that of a mainstream GNN system DGL [23]. We can conclude that the design of Hyperion keeps the model convergence.

Pre-processing Cost. We examine the pre-processing cost of two-hop GraphSAGE on the IG dataset. We set CPU/GPU cache memory to 200GB/40GB. The pre-processing only needs fewer than 20 seconds and thus can be tolerable, because each training epoch takes 19 seconds, and training a model often takes tens/hundreds of epochs for convergence.

2 Calculation of Monetary Cost

Table 1 shows the estimation of the 5-year total cost of ownership (TCO) of a local machine/cluster. The overall TCO is estimated as Equation 1, where C_{mac}, C_{gpu}, C_{ssd} and C_{ele} represent the cost of a single host machine, GPU, and SSD, and the electricity per machine. N_{gpu}, N_{ssd}, and N_{mac} represent the number of GPUs and SSDs per host machine and the number of host machines. We provide the cost of each individual component. C_{mac} is estimated as $14,098 using the price of machine A collected from its provider [20]. For the electricity cost (C_{ele}), we estimate the electricity price to be 10 cents per kWh and the power consumption of a single machine to be 4,000 Watts, assuming the machine runs GPU workloads continuously in its lifecycle ($17,000 in total). The prices of GPUs and SSDs are gathered from Amazon [2, 1]. The prices (C_{gpu}) of an 80GB A100 and H800 PCIe GPU are $14,177 and $44,000. The prices ($C_{ssd}$) of a P5510 and 980 pro SSD are $308 and $100. Though the absolute number of costs could vary over time and location, a single component, e.g., SSDs, can be much cheaper than an entire machine.

$$TCO = (C_{mac} + C_{gpu} \times N_{gpu} + C_{ssd} \times N_{ssd} + C_{ele}) \times N_{mac} \tag{1}$$

References

[1] Amazon. Intel D7-P5510 3.84 TB Solid State Drive from Amazon. https://www.amazon.sg/dp/B08R3XKK5H?ref_=mr_referred_us_sg_sg, 2024.

[2] Amazon. NVIDIA Tesla PCIe A100 from Amazon. https://www.amazon.sg/NVIDIA-Tesla-A100-Ampere-Graphics/dp/B08X13X6HF, 2024.

[3] Renzo Angles, János Benjamin Antal, Alex Averbuch, Altan Birler, Peter Boncz, Márton Búr, Orri Erling, Andrey Gubichev, Vlad Haprian, Moritz Kaufmann, Josep Lluís Larriba Pey, Norbert Martínez, József Marton, Marcus Paradies, Minh-Duc Pham, Arnau Prat-Pérez, David Püroja, Mirko Spasić, Benjamin A. Steer, Dávid Szakállas, Gábor Szárnyas, Jack Waudby, Mingxi Wu, and Yuchen Zhang. The ldbc social network benchmark. *arXiv preprint arXiv:2001.02299*, 2020.

[4] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 2004.

[5] Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. Bubing: Massive crawling for the masses. In *Proceedings of the 23th international conference on World Wide Web*, 2014.

[6] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*, pages 587–596, 2011.

[7] Paolo Boldi and Sebastiano Vigna. The web graph framework: Compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, 2004.

[8] Ping Gong, Renjie Liu, Zunyao Mao, Zhenkun Cai, Xiao Yan, Cheng Li, Minjie Wang, and Zhuozhao Li. gsampler: General and efficient gpu-based graph sampling for graph learning. 2023.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[10] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33, 2020.

[11] Intel. PCM. https://github.com/intel/pcm, 2022.

[12] Arpandeep Khatua, Vikram Sharma Mailthody, Bhagyashree Taleka, Tengfei Ma, Xiang Song, and Wen-mei Hwu. Igb: Addressing the gaps in labeling, features, heterogeneity, and size of public graph datasets for deep learning research. *arXiv preprint arXiv:2302.13522*, 2023.

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[14] Jeongmin Brian Park, Vikram Sharma Mailthody, Zaid Qureshi, and Wen-mei Hwu. Accelerating sampling and aggregation operations in gnn frameworks with gpu initiated direct storage accesses. *Proceedings of the VLDB Endowment*, 17(6):1227–1240, 2024.

[15] Yeonhong Park, Sunhong Min, and Jae W Lee. Ginex: Ssd-enabled billion-scale graph neural network training on a single machine via provably optimal inmemory caching. *Proceedings of the VLDB Endowment*, 15(11):2626–2639, 2022.

[16] QuiverTeam. Quiver. https://github.com/quiver-team/torch-quiver, 2021.

[17] Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, CJ Newburn, Dmitri Vainbrand, I-Hsin Chung, Michael Garland, William Dally, and Wen-mei Hwu. Gpu-initiated on-demand high-throughput storage access in the bam system architecture. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 325–339, 2023.

[18] Solidigm. D7-P5510 high-performing, standard-endurance PCIe 4.0 NVMe SSD drive. https://www.solidigm.com/products/data-center/d7/p5510.html, 2021.

[19] Jie Sun, Li Su, Zuocheng Shi, Wenting Shen, Zeke Wang, Lei Wang, Jie Zhang, Yong Li, Wenyuan Yu, Jingren Zhou, and Fei Wu. Legion: Automatically pushing the envelope of multi-gpu system for billion-scale gnn training. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 165–179, 2023.

[20] Supermicro. Supermicro 4U GPU SuperServer. https://store.supermicro.com/us_en/4u-gpu-superserver-as-4125gs-tnrt2.html, 2024.

[21] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[22] Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. Mariusgnn: Resource-efficient out-of-core training of graph neural networks. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 144–161, 2023.

[23] Minjie Yu Wang. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*, 2019.

[24] Jianbang Yang, Dahai Tang, Xiaoniu Song, Lei Wang, Qiang Yin, Rong Chen, Wenyuan Yu, and Jingren Zhou. Gnnlab: A factored system for sample-based gnn training over gpus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022.

[25] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, and George Karypis. Distributed hybrid cpu and gpu training for graph neural networks on billion-scale heterogeneous graphs. In *Proceedings of the 28th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2022.