FuseSampleAgg: One-Pass Neighborhood Estimation for Budgeted Knowledge-Graph Refresh and Validation

Aleksandar Stanković[1] [0009-0003-5238-7251] ✉, Haoran Du[2] [0009-0000-6542-6940], and Xinming Wang[3] [0009-0001-9111-4817]

[1] University of Novi Sad, Novi Sad, Serbia
stankovic.sv25.2022@uns.ac.rs
[2] Shanghai Key Lab of Intelligent Information Processing,
College of Computer Science and Artificial Intelligence, Fudan University, China
hrdu24@m.fudan.edu.cn
[3] Institute of Automation, Chinese Academy of Sciences, China
wangxinming2024@ia.ac.cn

Abstract. Operational knowledge-graph (KG) pipelines in networking and cybersecurity increasingly need to refresh embeddings under strict time, memory, and audit budgets, especially as curated feeds and LLM-assisted extraction accelerate KG updates. A recurring per-step cost in mini-batch KG learning is neighborhood-context estimation: uniform neighbor sampling without replacement followed by mean aggregation. Common frameworks realize this estimator via sampled-subgraph (block) materialization and intermediate feature gathers, adding kernel launches, allocator pressure, and transient memory spikes. We present FuseSampleAgg, a fused PyTorch CUDA operator that samples neighbors and emits the sampled-neighborhood mean directly, avoiding explicit block construction while preserving GraphSAGE-mean semantics for the same sampled neighbor IDs. FuseSampleAgg supports seed-controlled sampling and optional saved-index replay for reproducible validation and regression testing. Across large-graph mini-batch workloads, FuseSampleAgg improves FP32 end-to-end step latency by 2.24×–3.48× over tuned DGL baselines and reduces transient GPU memory by up to 160× in our measurements. On OGB KG completion benchmarks, such as WikiKG2 and BioKG, FuseSampleAgg reduces step time and peak VRAM while matching ranking quality within seed variability, improving time-to-quality for budgeted KG refresh.

Keywords: Knowledge Graphs · Knowledge Verification and Validation · Knowledge-Based Systems in Cybersecurity · Large Language Models · Graph Neural Networks · GPU Acceleration

1 Introduction

Knowledge graphs (KGs) increasingly need repeated embedding refresh as curated feeds, schema edits, and LLM-assisted extraction accelerate graph updates in domains such as networking and cybersecurity [9,23,18,28,19,11,7]. In these settings, representation learning becomes a recurring *maintenance* task performed under fixed time, GPU-memory, and auditability budgets [25,17]. A persistent per-step bottleneck is neighborhood-context estimation: GraphSAGE-style pipelines sample neighbors, materialize sampled subgraphs into blocks, gather features, and then aggregate, even though explicit block construction is not required by the estimator semantics [8,21,6].

We present FuseSampleAgg, a fused CUDA operator that jointly performs uniform neighbor sampling without replacement and mean aggregation, directly emitting the aggregated context for 1–2 hop encoders while avoiding explicit block materialization. The operator preserves GraphSAGE-mean semantics for fixed sampled neighbor IDs, supports seed-controlled sampling and optional saved-index replay for reproducible validation, and leaves the downstream encoder/decoder unchanged. Across large-graph mini-batch workloads and OGB KG completion benchmarks, FuseSampleAgg improves end-to-end step time and reduces transient GPU memory while preserving task metrics within expected stochastic variability.

The paper makes four contributions. First, it frames KG representation learning as a *budgeted refresh* workload where time-to-quality and memory stability are first-order constraints. Second, it formalizes a block-free realization of the sampled-neighborhood mean while preserving GraphSAGE-mean semantics for fixed sampled neighbor IDs. Third, it implements the fused operator as a PyTorch CUDA extension with determinism and optional replay support. Fourth, it evaluates runtime, memory, and task-level behavior on large-graph mini-batch workloads and OGB KG completion benchmarks.

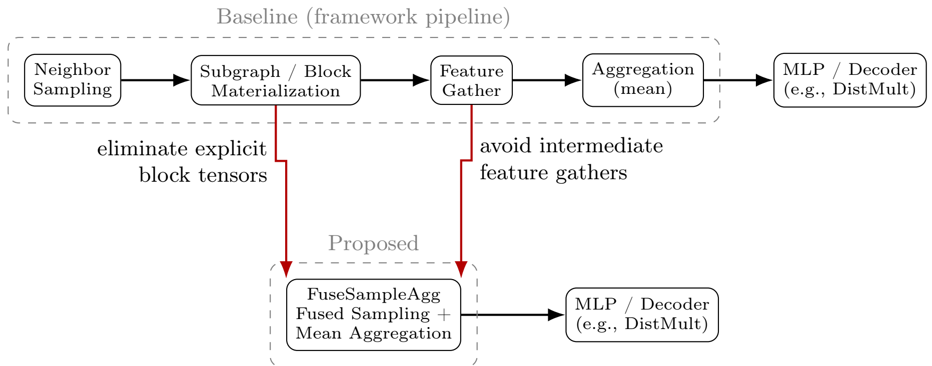

Fig. 1. Estimator realization in budgeted KG refresh. Standard pipelines often realize neighborhood-context estimation by materializing a sampled subgraph and then aggregating. FuseSampleAgg computes the same sampled-neighborhood mean estimator *for the same sampled neighbor IDs* directly, avoiding intermediate materialization that is not required by the estimator semantics.

2 Related Work

KG representation learning is widely used for completion, validation, and downstream knowledge services, while KG lifecycle work emphasizes continuous monitoring, reproducibility, and governance during maintenance [9,22,25,17]. In cybersecurity settings, KGs commonly integrate STIX/TAXII-style threat intelligence and increasingly incorporate LLM-assisted extraction, which further raises refresh frequency [1,16,14,11,7]. For scalable training, GraphSAGE-style systems sample bounded neighborhoods and aggregate them, usually through frameworks such as DGL or PyG that materialize sampled blocks before message passing [8,21,6]. Related systems accelerate sampling, loading, or sparse compute through caching, partitioning, optimized loaders, or fused message-passing kernels (e.g., NextDoor, GNNAdvisor, GraphBolt, fuseGNN, TCGNN) [12,24,5,3,13]; orthogonal work changes the sampling policy or estimator itself [2,29,27,4].

Where FuseSampleAgg Fits. FuseSampleAgg is complementary to these systems: it preserves the sampling policy and model form, but moves the fusion boundary earlier by computing the sampled-neighborhood mean *without* explicit block construction. This choice also motivates seed-controlled sampling and optional saved-index replay for repeatable validation in maintenance workflows.

Scope Relative to Other Accelerated Graph Frameworks. GraphBolt, fuseGNN, and TCGNN are important related systems, but they optimize different parts of the pipeline or assume different execution boundaries. GraphBolt primarily improves sampling and data movement, while fuseGNN and TCGNN focus on fused message passing or sparse compute after graph structure is already available. Our comparison protocol therefore uses tuned end-to-end DGL pipelines as the primary block-materializing baseline for the exact estimator studied here. Accordingly, we do not claim universal dominance over every GPU graph framework; rather, we show that moving the fusion boundary earlier yields consistent gains against a strong tuned production baseline.

3 Method

3.1 Problem Setting and Motivating Application

We study KG representation learning as a recurring *budgeted refresh* operation. As facts, schemas, and extraction rules evolve, representations must be refreshed under fixed time, memory, and auditability constraints. Let $Q(\theta; G)$ denote a downstream validation utility on snapshot G (e.g., validation accuracy or MRR), and define the lifecycle metric

$$T(\tau) = \inf\{\, t \;:\; Q(\theta_t; G^{(t)}) \geq \tau \,\},$$

which measures how quickly a refresh run reaches a target validation regime. This viewpoint motivates optimizing the per-step neighborhood-context estimator. A representative application is threat-intelligence KG maintenance,

where STIX/TAXII-compatible sources and LLM-assisted extraction accelerate updates and tighten validate→repair→retrain loops under shared GPU budgets [1,16,14,15]. In such settings, the bottleneck is often the sampling–materialization–aggregation path rather than dense tensor math.

3.2 Fused Sampling and Mean Aggregation

Let $G = (V, E)$ be stored in CSR $(\texttt{rowptr}, \texttt{col})$ with features/embeddings $X \in \mathbb{R}^{|V| \times D}$. Given seed nodes S and fanout k, GraphSAGE-mean uses the sampled-neighborhood mean

$$h_{\text{nei}}(v) = \frac{1}{t} \sum_{u \in U(v)} X[u], \qquad t = \min(k, |N(v)|),$$

where $U(v)$ is a uniform sample without replacement from $N(v)$ when $|N(v)| \geq k$. Framework baselines typically compute this by materializing sampled blocks and then aggregating.

FuseSampleAgg. FuseSampleAgg fuses sampling and mean aggregation in one CUDA operator: for each seed $v \in S$, it samples up to k neighbors from CSR and directly accumulates and writes $h_{\text{nei}}(v)$, avoiding explicit block tensors and intermediate gathered-feature tensors. For 2-hop context with fanouts (k_1, k_2), FuseSampleAgg computes the nested mean

$$\hat{X}_r = \frac{1}{|U_1(r)|} \sum_{u \in U_1(r)} \left(\frac{1}{|W(u)|} \sum_{w \in W(u)} X[w] \right),$$

skipping empty neighborhoods. For fixed sampled neighbor IDs, FuseSampleAgg matches the arithmetic result of block-based aggregation; and under uniform sampling without replacement, the sample mean is an unbiased estimator of the full neighbor mean [20].

Extension Feasibility Beyond Uniform Mean Aggregation. The same early-fusion idea extends directly to weighted averages by streaming weighted numerators and denominators without explicit blocks. Attention-style aggregation is more demanding because it requires online score computation and numerically stable normalization within each sampled neighborhood, and deeper-hop variants increase memory traffic, synchronization, and floating-point error accumulation. We therefore treat relation-aware, attention-based, and deeper-hop operators as natural but unvalidated extensions of the present design.

3.3 Semantics and Estimator Properties

Estimator-Level Semantic Equivalence (Fixed Sampled IDs). Fix a CSR graph, a seed set, and sampled neighbor indices. Then FuseSampleAgg emits the same arithmetic means as a baseline that (i) materializes the same sampled blocks/-subgraph and (ii) applies mean aggregation over that materialization.

Proposition 1 (Semantic equivalence for fixed sampled indices). *For fixed sampled neighbor IDs, the neighbor-mean outputs emitted by FuseSampleAgg are identical to those obtained by block materialization followed by GraphSAGE-mean aggregation.*

Proof (Sketch of proof). Both realizations compute the same mean over the same feature vectors; materialization changes only the intermediate representation of the sampled neighbor IDs.

Unbiasedness of the Sampled Neighbor Mean. When $|N(v)| \geq k$ and $U(v)$ is a uniformly random k-subset of $N(v)$ sampled without replacement, the sample mean is an unbiased estimator of the full neighbor mean.

Lemma 1 (Unbiased neighbor-mean estimator). *Assume $|N(v)| \geq k$ and $U(v)$ is a uniformly random k-subset of $N(v)$. Then*

$$\mathbb{E}\left[\frac{1}{k}\sum_{u\in U(v)} X[u]\right] = \frac{1}{|N(v)|}\sum_{u\in N(v)} X[u].$$

Proof (Sketch of proof). For each $u \in N(v)$, $\Pr[u \in U(v)] = k/|N(v)|$; linearity of expectation gives the result.

Reproducibility for Validation Workflows. FuseSampleAgg supports seed-controlled sampling to produce repeatable sampled neighbor IDs for a fixed CSR graph and seed/frontier order. Optionally, `save_indices` records sampled IDs in forward and replays them in backward, so gradients correspond to the same sampled neighborhoods. This enables controlled ablations and regression testing in continuous KG maintenance pipelines.

4 Implementation Summary

FuseSampleAgg is packaged as a PyTorch CUDA extension and operates on contiguous CSR (`int32`) and contiguous feature tensors. The 1-hop kernel uses a warp-per-seed mapping with shared-memory buffering for sampled indices; the 2-hop kernel uses a block-per-root mapping with small shared buffers for U and W. By design, no block tensors are materialized, reducing allocator pressure and intermediate memory traffic.

Complexity. For batch size B and feature width D: 1-hop requires $\Theta(B\,k\,D)$ feature loads and $\Theta(B\,D)$ writes, while 2-hop requires $\Theta(B\,k_1 k_2\,D)$ loads and $\Theta(B\,D)$ writes. In contrast to block-based baselines, FuseSampleAgg avoids constructing and traversing intermediate block structures, which drives the observed step-time and memory improvements.

5 Experimental Setup

Datasets. We evaluate on Reddit, ogbn-arxiv, and ogbn-products for large-graph mini-batch training, and on ogbl-wikikg2 and ogbl-biokg for KG completion [10].

Baselines. FuseSampleAgg is compared against DGL pipelines that sample, materialize blocks, and aggregate [21]; for fairness, we tune three representative DGL modes (GPU graph, CPU workers, CPU+UVA) and report the best per configuration.

Precision and metrics. All experiments use FP32. We report CUDA-synchronized end-to-end step latency (forward+backward+optimizer) together with *peak transient GPU memory* observed during the timed region, excluding persistent model parameters and resident feature storage. KG completion uses a lightweight 2-hop encoder composed with an MLP and DistMult [26] and reports MRR with the OGB evaluator.

Measurement details and fairness. We precompute a single CSR adjacency for each dataset and reuse it across baselines; unless stated otherwise, we use 20 warmup iterations and time 200 steps, reporting the median latency. Experiments run on a single NVIDIA A800-SXM4-40GB GPU with PyTorch CUDA, DGL, and OGB utilities; scripts that produce raw logs and tables are available at `https://github.com/SV25-22/FuseSampleAgg`.

6 Results

6.1 Task-Level Sanity Check: Quality Preservation and Time-to-Quality

A systems optimization is only useful for KG maintenance if it preserves task quality. In an architecturally matched PyG parity check of the same shallow 2-hop mean encoder, FuseSampleAgg shows only small final-accuracy differences that are consistent with sampling stochasticity and floating-point reduction order: on Reddit, test accuracy is 0.9473 versus 0.9491 while wall-clock drops from 3.8 s to 1.1 s; on ogbn-products, test accuracy is 0.7114 versus 0.7128 while wall-clock drops from 1.3 s to 0.9 s. The practical takeaway is improved *time-to-quality*: FuseSampleAgg reaches the same validation regime earlier under the same compute budget.

Table 1. Accuracy parity summary (final epoch). **Wall(s)** is the end-to-end wall-clock time to complete the run under the parity setup (same model and hyperparameters), measured on the same system.

Dataset	Variant	Train	Val	Test	Wall(s)
ogbn-products	pyg	0.8848	0.8750	0.7128	1.3
ogbn-products	fsa	0.8861	0.8738	0.7114	0.9
reddit	pyg	0.9613	0.9503	0.9491	3.8
reddit	fsa	0.9614	0.9486	0.9473	1.1

These remaining quality gaps are small and consistent with expected stochasticity from sampling streams and floating-point reduction order rather than a systematic degradation from early fusion. This is important for maintenance-oriented workloads: the systems gain is useful precisely because it does not come with a material task-quality penalty.

6.2 Budgeted Refresh Efficiency: Runtime and Transient Memory of Neighborhood Estimation

We next evaluate the practical effect of eliminating explicit subgraph materialization. Each step includes neighborhood estimation, forward/backward, and an optimizer update. For each (dataset, fanout, batch) point, we evaluate three representative DGL modes (`dgl_gpu`, `dgl_cpu_workers`, `dgl_cpu_uva`) and compare FuseSampleAgg against the *best* DGL mode by median step time, approximating a tuned practitioner baseline.

Table 2. FP32 end-to-end training-step latency (forward+backward+optimizer, CUDA-synchronized) and peak transient GPU memory during the timed region. Best DGL baseline is selected per configuration among GPU-graph, CPU-workers, and CPU+UVA modes. Speedup is (DGL ms / FSA ms), so values above 1 mean FuseSampleAgg is faster.

Dataset	Fanout	Batch	Best DGL	DGL ms	FSA ms	Speedup	DGL peak(MB)	FSA peak(MB)
ogbn-arxiv	15 10	512	dgl_cpu_uva	2.402	0.846	2.84	25.2	25.2
ogbn-arxiv	15 10	1024	dgl_cpu_uva	2.344	0.846	2.77	27.3	25.2
ogbn-arxiv	25 10	512	dgl_gpu	2.395	0.844	2.84	56.6	25.2
ogbn-arxiv	25 10	1024	dgl_gpu	2.419	0.896	2.70	75.5	25.2
ogbn-products	15 10	512	dgl_cpu_uva	2.258	0.847	2.66	29.4	23.1
ogbn-products	15 10	1024	dgl_cpu_uva	2.278	0.845	2.69	33.6	25.2
ogbn-products	25 10	512	dgl_cpu_uva	2.255	0.878	2.57	31.5	23.1
ogbn-products	25 10	1024	dgl_gpu	2.259	0.943	2.40	4028.6	25.2
reddit	15 10	512	dgl_gpu	3.696	1.438	2.57	3745.5	125.8
reddit	15 10	1024	dgl_gpu	5.438	1.563	3.48	3751.8	125.8
reddit	25 10	512	dgl_gpu	4.630	2.066	2.24	3749.7	125.8
reddit	25 10	1024	dgl_gpu	6.547	2.218	2.95	3756.0	125.8

Table 2 shows that FuseSampleAgg is faster than the best tuned DGL configuration in all reported settings, with FP32 speedups of 2.24×–3.48×. When GPU-resident block materialization is fastest, DGL can show multi-GB transient peaks, whereas FuseSampleAgg maintains a small and stable footprint by emitting aggregated features directly; when CPU+UVA is strongest, FuseSampleAgg

still improves latency by removing the remaining materialization and gather overheads. Operationally, lower transient memory and faster steps reduce OOM risk and make fixed-window refresh runs more practical.

6.3 Knowledge-Graph Completion: Step-Time, Memory, and Time-to-MRR

We also evaluate FuseSampleAgg on `ogbl-wikikg2` and `ogbl-biokg` [10] using the same lightweight 2-hop encoder and DistMult decoder across methods; the only difference is whether the 2-hop nested-mean context is computed through block materialization or through the fused operator.

Table 3 reports FP32 training-step latency and peak VRAM (mean±std over 5 seeds). On BioKG, FuseSampleAgg improves step time by 1.44×–1.86× and substantially reduces transient GPU memory (e.g., 738.9 MB → 392.3 MB at batch 1024), consistent with block construction being a large fraction of per-step cost. On WikiKG2, the speedup is smaller but consistent (1.15×–1.16×) and peak VRAM drops by about 6%, which is expected because the end-to-end step is more dominated by large embedding tables and KG scoring.

Table 3. KG completion (FP32): training-step latency and peak VRAM (mean±std over 5 seeds) for a DGL 2-hop baseline that materializes sampled neighborhoods into blocks versus FuseSampleAgg which fuses 2-hop sampling and nested-mean aggregation. Speedup is (DGL ms / FSA ms).

Dataset	Fanout	Batch	DGL ms	FSA ms	Speedup	DGL peak(MB)	FSA peak(MB)
ogbl-biokg	15 10	512	7.177 ± 0.080	3.867 ± 0.052	1.86	738.7 ± 0.0	306.7 ± 0.0
ogbl-biokg	15 10	1024	7.764 ± 0.064	4.583 ± 0.053	1.69	738.9 ± 0.0	392.3 ± 0.0
ogbl-biokg	25 10	512	7.223 ± 0.069	4.608 ± 0.067	1.57	738.7 ± 0.0	306.7 ± 0.0
ogbl-biokg	25 10	1024	7.780 ± 0.057	5.403 ± 0.065	1.44	738.9 ± 0.0	392.3 ± 0.0
ogbl-wikikg2	15 10	512	85.776 ± 0.078	73.862 ± 0.036	1.16	13170.7 ± 0.4	12369.2 ± 0.0
ogbl-wikikg2	15 10	1024	86.637 ± 0.099	74.647 ± 0.034	1.16	13193.3 ± 0.4	12374.9 ± 0.0
ogbl-wikikg2	25 10	512	85.839 ± 0.046	74.464 ± 0.031	1.15	13181.2 ± 0.6	12369.2 ± 0.0
ogbl-wikikg2	25 10	1024	86.745 ± 0.054	75.355 ± 0.038	1.15	13210.3 ± 0.5	12374.9 ± 0.0

Table 4 reports WikiKG2 MRR after 6000 training steps (mean±std over 5 seeds) at batch 1024. FuseSampleAgg matches the DGL baseline within run-to-run variability and is slightly higher in our measurements at fanout 25–10. Since WikiKG2 step time drops from 86.704 ms to 75.960 ms, 6000 steps finish about 12% faster; equivalently, within the wall-clock time required by DGL to run 6000 steps, FuseSampleAgg can execute about 6850 steps. Thus the systems gain translates directly into improved *time-to-quality* for repeated KG refresh.

Table 4. WikiKG2 final MRR after 6000-step training (mean±std over 5 seeds).

Fanout	Variant	Batch	#Seeds	Step ms (mean±std)	Valid MRR (mean±std)	Test MRR (mean±std)
15 10	dgl	1024	5	86.619 ± 0.031	0.2284 ± 0.0087	0.2568 ± 0.0074
15 10	fsa	1024	5	75.097 ± 0.018	0.2293 ± 0.0100	0.2592 ± 0.0100
25 10	dgl	1024	5	86.704 ± 0.067	0.2275 ± 0.0101	0.2570 ± 0.0084
25 10	fsa	1024	5	75.960 ± 0.045	0.2291 ± 0.0111	0.2586 ± 0.0102

The KG results also clarify where the method helps most. On BioKG, where the neighborhood estimator is a larger fraction of end-to-end step cost, the gains are larger in both runtime and transient memory. On WikiKG2, the operator still helps, but the step includes heavier embedding-table and scoring work, so the systems improvement is diluted by other dominant costs. This pattern is consistent with the paper's main claim: removing block materialization is most valuable when that path is itself a major bottleneck.

7 Threats to Validity and Limitations

Our goal is to accelerate the sampler→materialize→aggregate path for a widely used 1–2 hop GraphSAGE-mean primitive; we do not claim state-of-the-art task accuracy or a complete cross-framework leaderboard. Relative performance depends on hardware, runtime behavior, and which baseline mode is strongest for a given platform. Our baselines cover tuned DGL execution modes but not every accelerated graph framework, and our experiments are FP32-only, so the present paper does not establish behavior under FP16/BF16 training. Task metrics remain stochastic because of neighbor sampling, negative sampling, and floating-point reduction order, although FuseSampleAgg supports saved-index replay for controlled comparisons. Finally, the KG completion encoder is intentionally lightweight and does not capture relation-aware, attention-based, or deeper-hop message passing; extending early fusion to richer operators is a natural direction for future work.

8 Conclusion

We presented FuseSampleAgg, a block-free realization of the common neighborhood-context estimator formed by uniform neighbor sampling without replacement followed by mean aggregation. By preserving estimator semantics while avoiding intermediate block materialization, FuseSampleAgg improves time-to-quality under fixed refresh budgets and supports reproducible validation via seed control and optional replay [25,17]. Across large-graph mini-batch workloads and KG completion benchmarks, FuseSampleAgg consistently reduces end-to-end step time and transient GPU memory while preserving task metrics within expected stochastic variability. These results suggest that earlier-fusion systems primitives can make repeated KG refresh and validation substantially more practical. More broadly, they support the design principle that moving the fusion boundary earlier can be valuable even when the downstream model class remains unchanged.

Acknowledgments. The authors thank the anonymous reviewers for their constructive feedback.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Stix version 2.1. Tech. rep., OASIS Cyber Threat Intelligence (CTI) TC (Feb 2021), https://docs.oasis-open.org/cti/stix/v2.1/stix-v2.1.html, oASIS Standard
2. Chen, J., Ma, T., Xiao, C.: Fastgcn: Fast learning with graph convolutional networks via importance sampling. In: International Conference on Learning Representations (ICLR) (2018)
3. Chen, X., Yan, M., et al.: Fusegnn: Accelerating graph convolutional neural network training on gpgpu. In: ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC) (2021)
4. Chiang, W.L., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.J.: Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) (2019)
5. DGL Team: Graphbolt: High-throughput dgl dataloading and sampling for gnns (2024), project documentation
6. Fey, M., Lenssen, J.E.: Fast graph representation learning with pytorch geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
7. Fieblinger, T., Kacianka, S., Steineder, A., Lamprecht, D.: Actionable cyber threat intelligence using knowledge graphs and large language models (2024), https://arxiv.org/abs/2407.01822
8. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems (NeurIPS) (2017)
9. Hogan, A., Blomqvist, E., Cochez, M., Gutierrez, C., Kirrane, S., Gayo, J.E.L., Navigli, R., Polleres, A., Szekely, S., et al.: Knowledge graphs. ACM Computing Surveys (2022)
10. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., Leskovec, J.: Open graph benchmark: Datasets for machine learning on graphs. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)
11. Hu, Y., Zou, F., Han, J., Sun, X., Wang, Y.: Llm-tikg: Threat intelligence knowledge graph construction utilizing large language model. Computers & Security **145**, 103999 (2024). https://doi.org/10.1016/j.cose.2024.103999
12. Jangda, A., Kamburugamuve, S., Sergey, I., Guha, A.: Accelerating graph sampling for graph machine learning with nextdoor. In: European Conference on Computer Systems (EuroSys) (2021)
13. Kao, S., Sukumaran-Rajam, A., et al.: Tc-gnn: Bridging sparse gnns and dense tensor cores on gpus. In: HPDC (2023)
14. MITRE: attack-stix-data: Mitre att&ck in stix 2.1. https://github.com/mitre/attack-stix-data (2025), accessed 2025-12-29
15. MITRE: MITRE ATT&CK: Data and tools (stix/taxii). https://attack.mitre.org/resources/ (2025)
16. OASIS Open: TAXII version 2.1. https://docs.oasis-open.org/cti/taxii/v2.1/taxii-v2.1.html (2021), oASIS Standard
17. Pellegrino, M., Tuozzo, G., Rula, A.: KGHeartBeat: A community-shared open-source tool for knowledge graph quality assessment. In: Extended Semantic Web Conference (ESWC) – Demo/Poster Track (2024)
18. Sikos, L.F.: Cybersecurity knowledge graphs. Knowledge and Information Systems (2023). https://doi.org/10.1007/s10115-023-01860-3
19. Sun, N., Ding, M., et al.: Cyber threat intelligence mining for proactive cybersecurity defense: A survey and new perspectives. IEEE Communications Surveys & Tutorials (2023). https://doi.org/10.1109/COMST.2023.3273282

20. Vitter, J.S.: Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) **11**(1), 37–57 (1985)
21. Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Ma, Q., et al.: Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
22. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering **29**(12), 2724–2743 (2017)
23. Wang, X., Xu, J., Feng, A.H., Chen, Y., Guo, H., Zhu, F., Shao, Y., Ren, M., Yi, H., Lian, S., et al.: The hitchhiker's guide to autonomous research: A survey of scientific agents. TechRxiv.August 07, 2025. DOI:10.36227/techrxiv175459840.02185500/V1 (2025)
24. Wang, Y., Feng, B., Li, G., Li, S., Deng, L., Xie, Y., Ding, Y.: Gnnadvisor: An adaptive and efficient runtime system for gnn acceleration on gpus. In: USENIX Symposium on Operating Systems Design and Implementation (OSDI) (2021)
25. Xue, B., Zou, L.: Knowledge graph quality management: A comprehensive review. IEEE Transactions on Knowledge and Data Engineering **35**(5), 4969–4988 (2023). `https://doi.org/10.1109/TKDE.2022.3150080`
26. Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: International Conference on Learning Representations (ICLR) (2015)
27. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: Graphsaint: Graph sampling based inductive learning method. In: International Conference on Learning Representations (ICLR) (2020)
28. Zhao, X., Jiang, R., Han, Y., Li, A., Peng, Z.: A survey on cybersecurity knowledge graph construction. Computers & Security **136**, 103524 (2024). `https://doi.org/10.1016/j.cose.2023.103524`
29. Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., Gu, Q.: Layer-dependent importance sampling for training deep and large graph convolutional networks. In: Advances in Neural Information Processing Systems (NeurIPS) (2019)