*algorithms*

MDPI

*Article*

# SoftPart GNN: Distributed Graph Neural Network Training Without Offline Graph Partitioning

**Attila Farkas [1,2,*], Mátyás Nagy [1,2] and Gábor Kertész [1,2]**

[1] Laboratory of Parallel and Distributed Systems, Institute for Computer Science and Control (SZTAKI), Hungarian Research Network (HUN-REN), 1111 Budapest, Hungary; matyas.nagy@sztaki.hun-ren.hu (M.N.); kertesz.gabor@nik.uni-obuda.hu (G.K.)
[2] John von Neumann Faculty of Informatics, Obuda University, 1034 Budapest, Hungary
* Correspondence: attila.farkas@sztaki.hun-ren.hu

**Abstract**

Distributed graph neural network training usually starts with an offline partitioner such as METIS, which fixes the layout before any workload is seen. SoftPart GNN is a distributed GraphSAGE runtime on PyTorch DistributedDataParallel (DDP) without one. A short online warm-up turns the vertices that the training sampler touches into an exclusive soft ownership, and a budgeted cache replicates the context that it omits. Training samples only from this local view, so gradient synchronisation is the only per-step cross-worker traffic. A layout-independent full-neighbour evaluator exchanges hidden representations in place of features and, on the four graphs where we verified it, agrees with a single-machine reference to rounding level. We measured a two-layer GraphSAGE of hidden width 16 on OGBN-Arxiv, Reddit, OGBN-Products, IGB-hom-small and IGB-hom-medium at four to 16 nodes with one GPU each, and on four graphs at 32. At four to 16 nodes, SoftPart GNN completes training, placement included, in 22 to 62 percent of the time of a DistDGL baseline on the Deep Graph Library's (DGL) original distributed sampler, and 21 to 52 percent with the baseline's partitioning counted. Its test accuracy there stays within 0.006 of DistDGL's on four of the five graphs. At 32 nodes, the ratio is 0.40 to 0.91. On Reddit, the accuracy is 0.0042 lower, and OGBN-Arxiv is left open. A paired control with modulo ownership and the same cache shows no detected accuracy difference. At four to 16 nodes, the warm-up shortens the training epoch on four graphs and gives no end-to-end gain.

**Keywords:** distributed graph neural network training; graph partitioning; partitioner-free training; GraphSAGE; feature cache; node ownership; exact distributed evaluation; DistDGL; high-performance computing

Check for updates

Academic Editor: Guanqiu Qi

Received: 20 September 2026
Revised: 9 October 2026
Accepted: 16 October 2026
Published: 21 October 2026

**Copyright:** © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

## 1. Introduction

Distributed training of graph neural networks (GNNs) at the scale where a single device no longer suffices almost universally begins with a step that is not training at all. An offline partitioner such as METIS cuts the graph into a fixed number of partitions, and that layout decides which worker stores and samples which vertices for the entire run [1–3]. A high-quality partition reduces cross-worker references, but producing it is a separate, memory-intensive workload, and the layout is fixed before any sampling has been observed. Pipelined push–pull training [4], streaming partitioning [5], the split-parallel training with a lightweight probabilistic partitioning algorithm of Spa (preprint GSplit) [6]

and partitioner replacement [7] all name this cost as their motivation, and comparative studies measure it directly [8]. $P^3$ already removes the partitioner by placing vertices with a hash, which moves the cost into per-layer feature traffic during training. The open question is whether a placement derived from the training workload can remove the offline step and still keep communication out of the training step (Section 2.3). The graphs measured here range from 0.17 to 10 M vertices, and on the four smaller ones, the feature matrix fits one contemporary accelerator, so the comparison is between two distributed systems (Section 5).

We answer that question with SoftPart GNN, a distributed GraphSAGE training runtime on PyTorch 2.8 and its `DistributedDataParallel` implementation that runs no offline partitioner. We call the design *partitioner-free* in a specific sense. No vertex-to-worker layout is pre-computed before training has been observed, and no cut or balance objective is ever optimised. Instead, the layout is a by-product of sampling the training workload once, in a short warm-up, and is then fixed for the run. We avoid the term *partition-free*, since exclusive ownership is itself a partition of the training seeds (Section 3.6). Figure 1 shows the pipeline. A short online *locality warm-up* samples the training workload with the fan-outs training will later use, and a deterministic resolution turns the vertices that each worker touched into an exclusive *soft ownership*. A memory-budgeted *foreign-vertex cache* then replicates the neighbourhood context that ownership leaves out. Ownership decides which worker trains a vertex and caching decides what is available locally, and we keep the two decisions separate on purpose. Training samples exclusively from the worker's local view, with no per-minibatch remote graph fetch, so DDP gradient synchronisation is the only per-step cross-worker traffic. We compute validation and the final test with an exact full-neighbour evaluator that is independent of the training layout (Section 4), because a locality-restricted training view would confound model quality with placement quality.

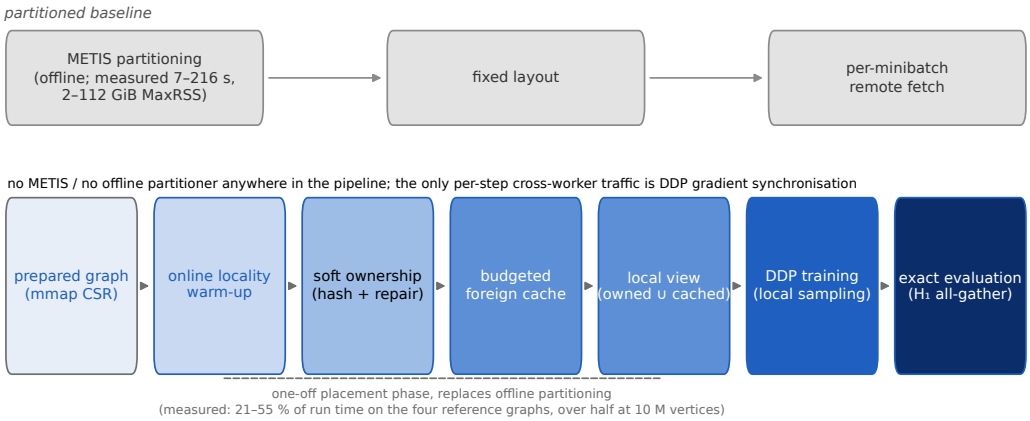



**Figure 1.** SoftPart GNN workflow. All workers share one mmap-able graph store. A one-off placement phase (warm-up → ownership → cache → local view) replaces offline partitioning. Training samples from the local view only. Validation and testing use the exact evaluator of Section 4.

SoftPart GNN gives up partition quality, and guarantees neither a minimal edge cut, perfect balance, nor unchanged training samples. In exchange, it obtains a placement whose cost is not negligible (Section 6.2), budgeted replication, and bulk-only communication.

Our main contributions are as follows:

1. A partitioner-free design in which a workload-driven warm-up and a degree/shortfall-scored cache replace the offline partitioner, with training responsibility (ownership) and data availability (cache) as separate decisions (Section 3), and with an explicit statement of its guarantees and non-guarantees (Section 3.2). A paired control replaces only the ownership map by $o(v) = v \bmod W$ and keeps the cache policy and budget

fixed. It shows no detected accuracy difference within the seed spread (Section 6.6), so the measurements isolate the ownership decision and leave the cache's own contribution unseparated (Section 7). Against that control, the warm-up shortens the training epoch on four of the five graphs at four to 16 nodes and, where ownership coverage is partial, leaves fewer resident owner rows, both paid for by placement time. That saving is part of the controlled approximation, so Section 6.6 does not count it as a separate gain. The warm-up gives no end-to-end gain of its own on any graph and buys no higher accuracy. At ten million vertices and at 32 workers, its placement cost outweighs the shorter epochs.

2. A full-neighbour evaluation protocol for two-layer mean GraphSAGE that does not depend on the training layout. It computes the hidden layer once on the global store and replicates it with a single narrow all-gather, so its output is a function of the prepared graph, the features, the parameters and the target set only (Section 4.2). The layer-wise decomposition itself is standard [9]. What we add is its use as a layout-independent measurement across systems, which we verified to the rounding level against a single-machine reference on four graphs and on models trained by both systems (Section 6.8). DGL's own full-neighbour inference scores the DistDGL baseline, and the two evaluators agree in what they compute while differing in how accuracy is aggregated (Section 5).
3. A measurement against a DistDGL baseline in a fixed two-sampler configuration on DGL's original distributed sampler. It covers five node-classification graphs (OGBN-Arxiv, Reddit, OGBN-Products, IGB-hom-small, IGB-hom-medium) at four, eight and 16 nodes with one GPU each and five runs per cell on both sides, and the first four graphs at 32 nodes. At four to 16 nodes, the end-to-end time, from warm-up to the final test, is 0.22–0.62 of DistDGL's, and 0.21–0.52 with the baseline's partitioning counted, and test accuracy stays within 0.006 of DistDGL's on four of the five graphs, with OGBN-Arxiv left open. At 32 nodes, the ratio is 0.40–0.91, Reddit falls outside the 0.006 band, and OGBN-Arxiv is left open (Section 6).
4. A quantified account of the design's own cost. Placement takes 20.6–55.0% of SoftPart's internal end-to-end time on the four smaller graphs, and on IGB-hom-medium, the ownership stage alone takes 54–61%. Its share of the run grows with the world size on every graph, and on OGBN-Products, its time grows as well while training shrinks (Section 6.2). Among the measured components, the owner-map build, repeated at every coverage checkpoint, is the largest term on four of the five graphs (all but Reddit), and the all-gather claim exchange grows as the square of the world size at a fixed number of checkpoints (Section 3.10).

## 2. Related Work

### *2.1. Partition-Based Systems and the Partitioner-Free Endpoint*

The systems that this work positions itself against fix a vertex-to-worker layout before training. DistDGL [1] partitions with METIS, co-locates a graph server and trainers per partition, and fetches remote features through its RPC layer. AliGraph, ByteGNN, NeuGraph, Sancus and NeutronStar vary the storage, scheduling and communication strategy but keep the offline partition [2,3,10–12]. BETTY and HongTu apply the same partition-first approach within one multi-GPU server [13,14]. DSP samples over a partitioned graph on multiple GPUs with an interconnect-aware data layout and a collective sampling primitive [15]. We use DistDGL as the baseline because it is the reference implementation of this family.

$P^3$ [4] is the system that most directly takes over the partitioner's locality role. It hash-partitions vertices and then pipelines a push–pull execution that moves the first layer's computation to where the features live, so hash placement alone is only its starting

point. It replaces the partitioner with more communication and a model of intra-layer parallelism, whereas SoftPart GNN replaces it with less, by paying a one-off placement cost and then sampling locally, so the two systems are endpoints of the same design axis. $P^3$ is available only as a paper, and the systems that compare against it, among them Spa [6] and LeapGNN [16], work from the published description. Within a single machine, DiskGNN derives its on-disk feature layout from an offline pre-sampling pass [17], the same signal that this design uses for cross-machine ownership, which is an analogy of mechanism only, since the two scopes differ.

### *2.2. Cheaper Partitions, Caches and Replication*

Another line of work measures the partitioning step itself, or reduces its cost without removing it. Comparative studies quantify partitioner quality against training speed [8]. Spa (preprint GSplit) replaces data-parallel minibatch training with split parallelism and adds a probabilistic partitioner that minimises the communication that this introduces [6]. Armada's GREM partitioner streams edges to obtain a partition at a fraction of METIS's memory [7]. All three keep a partition and make it cheaper. Spa is also the closest precedent for using the sampler's own accesses as the placement signal, offline and for GPU placement within one host. Streaming partitioners assign vertices in one pass over the edge stream, in ASEPR after an attribute-driven clustering step [5], and online re-partitioning moves them while training proceeds [18], so the output is again a partition judged by cut quality. SoftPart's warm-up produces an ownership map, and the only judgement on it is whether the subsequent local sampling is sufficient.

A third group puts a cache in front of a partitioned or disk-resident store. PaGraph caches high out-degree vertices per GPU [19], while Ginex, DiskGNN and OUTRE move features between SSD, host and device [17,20,21], and MassiveGNN prefetches and evicts remote features across nodes on top of DistDGL [22]. BGL pairs a dynamic cache with an improved partitioner [23], Legion a multi-GPU cache hierarchy with a hierarchical partition [24], and WholeGraph holds structure and features in a node's distributed shared memory instead [25]. GNNLab's static GPU feature cache, the closest policy to `degree_shortfall`, is chosen from a pre-sampling pass over the training workload [26], so workload-derived cache selection has precedent. Those designs keep the partitioner and add a cache, whereas SoftPart drops the partitioner and keeps a cache, which is also its only mechanism that makes foreign vertices visible, since there is no fallback remote fetch.

Halo and ghost vertices, vertex-cut replication and overlapping partitions all replicate boundary state to reduce communication, and SALIENT++ combines replication with partitioning by selecting the frequently accessed out-of-partition rows through a vertex-inclusion-probability analysis [27]. We use the term *soft partition* for the specific combination of exclusive ownership with non-exclusive, budgeted availability, and boundary replication as such is not new.

Accuracy under locality-restricted sampling is the risk most directly relevant to this paper. A restricted neighbourhood degrades model quality, as demonstrated empirically for link prediction. Sharing the full graph with every worker removes the degradation at a high communication cost, and graph sparsification is proposed there as a cheaper approximation [28]. GNNAutoScale bounds the error of approximating with historical embeddings [29], and BNS-GCN subsamples each partition's boundary vertices at every iteration and quantifies the accuracy cost [30]. Those works restrict neighbourhoods inside a partitioned full-graph setting, whereas SoftPart's restriction is a by-product of a sampled trainer's local view and is measured with a layout-independent evaluator. SoftPart does not guarantee that training samples match unrestricted global sampling (Section 3.2). What

it guarantees is that the measurement of the resulting model is exact, so that any such degradation becomes visible (Section 4).

### *2.3. The Gap*

None of SoftPart's mechanisms are new on their own. Warm-up sampling, hash and modulo ownership, degree-ranked caches, boundary replication and layer-wise exact evaluation all have precedents cited above. The gap that we state is narrow. Among the sampled, data-parallel trainers that we screened, none let the observed training workload decide which worker trains which vertex, and none pair such an ownership with a cache that fills the context that the ownership omits while steady-state traffic stays at gradient synchronisation. Table 1 places the four closest systems against SoftPart on the four axes that separate them.

**Table 1.** The four closest systems and SoftPart GNN on the axes that separate them. "Workload" means the training sampler's observed accesses; "offline" means before training starts. Spa/GSplit and GNNLab place within one host, so they fall outside the cross-machine count of the text.

| System | Vertex-to-Worker Assignment | Offline Partitioner | Cache | Evaluation Independent of the Layout |
|---|---|---|---|---|
| $P^3$ [4] | hash, then push–pull moves the first layer to the features | none | none, features stay in place | no |
| ROC [31] | contiguous vertex ranges re-fitted each iteration to measured per-partition time, full-graph training | none | none, memory manager swaps to host | no |
| Spa/GSplit [6] | GPUs within one host, weighted min-cut on pre-run sampler access counts | yes, offline per host | replication implied by the split | no |
| GNNLab [26] | GPUs within one host, by the factored sampler/trainer roles | none | static GPU feature cache chosen by a pre-sampling pass | not stated |
| SoftPart GNN | exclusive soft ownership from an online warm-up of the training sampler, repaired to cover every seed | none | budgeted foreign-vertex cache by degree and shortfall, no remote fetch | yes, full-neighbour evaluator (Section 4) |

The corpus behind this section was harvested from OpenAlex, Crossref, DBLP and arXiv with keyword queries on distributed GNN training, the full proceedings of twenty systems and machine-learning venues, and a citation-snowball round, then screened on title and abstract. From the screened records and the systems cited above, we retained 49 named GNN training systems and labelled each from its full text or official documentation by how its training layout is produced. For every system, the evidence sentence and its location were recorded, and every system labelled as deciding ownership from the workload, together with every borderline case, was checked a second time against its full text. Of the 49, 20 are single-machine designs (multi-GPU or out-of-core) or inference-only systems with no cross-machine training layout. Of the 29 that train across machines, 11 obtain the layout from a cut-optimising partitioner, five from a streaming or online partitioner, eight from hash, random or range placement, two replicate the graph on every machine, and two do not state their partitioner. One system, ROC [31], derives its layout from the observed training workload, re-fitting contiguous vertex ranges each iteration to a cost model of measured per-partition execution time in full-graph training. Those ranges partition a full-graph workload, and the ownership that a sampled trainer needs is a different object.

DistDGL itself offers a member of the hash/random class, since its partitioning API accepts a `random` `part_method` alongside METIS [1], a layout that is METIS-free but still not derived from the training workload. Workload-derived availability does exist, in SALIENT++'s replication analysis [27] and in GNNLab's pre-sampling cache policy [26], so the gap concerns ownership only. Targeted vocabulary probes for *partition-free*, *partitioning-free* and *without pre-partitioning* returned no distributed GNN training system, and *soft partition* returned zero records in the titles and abstracts screened. The labelled list of the 49 systems is Supplementary Table S1. The premise, that offline partitioning is expensive, is shared by at least four independent works [4–7] and is not new.

## 3. System Design

A runtime without a partitioner still decides which worker trains each vertex and which vertices it holds locally, and SoftPart GNN separates the two decisions (Figure 1).

### *3.1. Notation and Terminology*

In what follows, the prepared graph is a directed graph $G = (V, E)$ held in compressed sparse row form, with $N = |V|$ vertices, $M = |E|$ edges and $\mathcal{N}(v)$ the in-neighbourhood that the sampler reads. Input features are $X \in \mathbb{R}^{N \times F}$ of width $F$, the hidden width is $D$, and the model has $L = 2$ layers with the neighbour fan-outs $(k_1, k_2)$ of GraphSAGE minibatch sampling [32], of which $k = k_2$ is the last one. Our reference configuration uses $(10, 25)$, $D = 16$ and a per-rank batch size $B = 512$ (Section 5). The $W$ workers are the ranks $w = 0, \ldots, W - 1$ of a PyTorch Distributed data-parallel group [33,34], $S \subseteq V$ is the set of training seeds, and $S_{\text{val}}$, $S_{\text{test}}$ are the validation and test target sets. Each vertex carries one of $C$ class labels (Table 2), and $H_1$ is the first-layer hidden representation of Equation (1) (Section 4.2). In the pseudocode, $o(v)$ is the owner of vertex $v$, $\perp$ marks an unassigned vertex, $o_{\text{pre}}$ is ownership before repair, and the fixed repair salt $s_r$ (fixed at $s_r = 12{,}345$ in the implementation, without a run parameter) separates the repair hash of Section 3.6 from the resolution hash.

**Table 2.** Datasets of the experiments. $F$ is the feature width. Edges are directed CSR entries of the prepared store, identical to the DistDGL graph. Splits are each dataset's standard split.

| Dataset | Vertices | Edges | $F$ | Classes |
|---|---|---|---|---|
| OGBN-Arxiv (bidir.) [1] | 169,343 | 2,315,598 | 128 | 40 |
| Reddit [2] | 232,965 | 114,848,857 | 602 | 41 |
| OGBN-Products [3] | 2,449,029 | 123,718,280 | 100 | 47 |
| IGB-hom-small [4] | 1,000,000 | 13,070,502 | 1024 | 19 |
| IGB-hom-medium [5] | 10,000,000 | 130,077,694 | 1024 | 19 |

[1] Reverse edges added to the citation graph, reciprocal duplicates removed, no self-loops; time split. [2] DGL's self-loop graph as provided, with both directions plus one self-loop per vertex; DGL node-type split. [3] Both directions of the co-purchase edges, no self-loops; sales-ranking split. [4] Released directed edges (12,070,502) plus one normalised self-loop per vertex; no reverse edges; released 60/20/20 split with 600,000 training, 200,000 validation and 200,000 test vertices. [5] As IGB-hom-small; 6,000,000 training, 2,000,000 validation and 2,000,000 test vertices.

Five terms recur throughout, all defined for a world size $W$ and a fixed seed $s$. The *claim set* $C_w$ of rank $w$ is the set of vertices that rank $w$ touched in any sampled layer during warm-up. It is a set, so repeated touches are recorded as diagnostics and do not weight resolution. *Ownership* assigns each vertex to at most one rank or leaves it unassigned (Invariant 1), and $O_w$ is the set of vertices owned by rank $w$. The *active scope* $V_{\text{act}}$ is the vertex set that must be owned once repair has run, which in the reference configuration is the set of training seeds (Invariant 2). The *cache* $K_w$ of rank $w$ is a set of vertices not owned

by $w$ (Invariant 3), that is, vertices owned by another rank or by none, $o(v) \neq w$ including $o(v) = \perp$, and the caches of different ranks may overlap (Invariant 4). The *local view* of rank $w$ is its available set $A_w = O_w \cup K_w$ together with the edges of $G$ whose both endpoints are available, and the training sampler on rank $w$ reads only this view and the feature rows of $A_w$ (Invariant 5, Section 3.8).

Placement diagnostics are reported in Section 6. *Warm-up coverage* $\gamma$ is the fraction of training seeds that have an owner before repair, the quantity that the stopping test of Algorithm 1 evaluates, and the *repaired fraction* of training seeds is $1 - \gamma$. The *graph-wide assignment* is the fraction of all $N$ vertices that have an owner once repair has run (Section 6.5). Since repair touches training seeds only, it exceeds the pre-repair value by at most $(1-\gamma)|S|/N$, at most 1.1 percentage points in the sweep. *Seed balance* is the coefficient of variation $\mathrm{CV}_w$ of owned training seeds across ranks.

**Algorithm 1** locality warm-up

**Input:** $G$, $S$, fan-outs $(k_1, k_2)$, batch size $B$, rank $w$, world size $W$, seed $s$, coverage target $\tau$ (0.98 in the reference configuration)

$\texttt{rng} \leftarrow \textsc{InitRng}(s + 17017)$ ▷ one stream, consumed sequentially below
$\texttt{order} \leftarrow \textsc{Shuffle}(S, \texttt{rng})$
**if** $\tau > 0$ **then**
  $\texttt{order} \leftarrow \texttt{order} \,\|\, \textsc{Shuffle}(V \setminus S, \texttt{rng})$ ▷ extra seeds appended, same stream continued, disjoint vertex set
**end if**
$\texttt{seeds}_w \leftarrow \texttt{order}[w :: W]$ ▷ round-robin share
$C_w \leftarrow \emptyset$
**for** each batch $\beta$ of $\texttt{seeds}_w$ **do**
  $\texttt{blocks} \leftarrow \textsc{SampleTwoLayer}(G, \beta, (k_1, k_2), \texttt{rng})$ ▷ reads the global CSR
  $C_w \leftarrow C_w \cup \{\text{all vertices of all sampled layers of } \texttt{blocks}\}$
  **if** $\beta$ is a coverage checkpoint **then**
    $o_{\text{pre}} \leftarrow \textsc{Resolve}(C_w, s)$ ▷ Algorithm A1
    $\gamma \leftarrow |\{v \in S : o_{\text{pre}}(v) \neq \perp\}|/|S|$
    **if** $\gamma \geq \tau$ **then**
      **stop**
    **end if**
  **end if**
**end for**
$o_{\text{pre}} \leftarrow \textsc{Resolve}(C_w, s)$ ▷ final resolution, also when the last batch was not a checkpoint
**return** $o_{\text{pre}}$ ▷ handed to Algorithm 2; $C_w$ is kept for diagnostics only

### *3.2. What the Runtime Guarantees*

The mechanisms of the subsections that follow are governed by nine invariants. The following properties hold in the reference implementation. Invariants 6–8 are enforced at runtime by a maximum all-reduce over the per-rank step counts, a global sum of correct and target counts and a configuration-key comparison before reuse. The others hold by construction of the placement and sampling path, and Invariant 9 is checked by the pipeline outside the runtime. Terminology is that of Section 3.1.

1. Every assigned vertex has at most one owner (by construction).
2. Every active training seed has an owner after the repair phase (Algorithm 2).

3. A cached vertex is never owned by the caching rank (by construction). Caching transfers availability, never ownership, which is the separation that a vertex cut makes between a replicated vertex's mirrors and its master [35].
4. A vertex may appear in the caches of several workers.
5. Every vertex sampled on the strict training path, which is the sampler and feature-gather path of the training loop as opposed to warm-up and evaluation, has a valid local feature row.
6. Every DDP rank executes the same number of backward collectives per epoch.
7. Exact evaluation aggregates global correct-prediction and vertex counts, acc $= \sum_w c_w / \sum_w n_w$ with $c_w$ correct predictions and $n_w$ targets on rank $w$, and never takes the mean of the per-rank ratios.
8. A cached first-layer representation ($H_1$) is reused only if both the model version and the configuration key match.
9. The prepared graph's semantics (direction, symmetrisation, self-loops) are identical for every system under comparison, so there is one prepared $G$ for all systems compared. All compared artifacts are generated from one prepared graph per dataset. SoftPart reads that store directly, and DistDGL reads partitions generated from it. Before each training job, the global vertex and edge counts of the DistDGL graph were checked against the store, and they match on all five graphs. A further check on one four-part partition per graph compares the SHA-256 of the sorted degree sequence, the self-loops, the labels, the splits and the feature sums, and all of them agree on all five graphs (Section 5.3).

**Algorithm 2** coverage repair, `active_neighbor_hash`

**Input:** $o$ (with $\perp$ entries), active scope $V_{\text{act}}$, $G$, seed $s$, repair salt $s_r$, world size $W$

`missing` $\leftarrow \{v \in V_{\text{act}} : o(v) = \perp\}$
**for** $v \in$ `missing` **do**
  $c_w \leftarrow |\{u \in \mathcal{N}(v) : o(u) = w\}|$ for every $w \in [W]$
  **if** $\max_w c_w > 0$ **then**
    $o(v) \leftarrow \arg\max_w c_w$ ▷ ties go to the lowest maximal rank
  **end if**
**end for**
`remaining` $\leftarrow \{v \in V_{\text{act}} : o(v) = \perp\}$
**for** $v \in$ `remaining` **do**
  $o(v) \leftarrow \text{HASH}(v, s, s_r) \bmod W$
  ▷ $\text{HASH}(v, s, s_r) = (1103515245\, v + s + s_r) \bmod 2^{31}$
**end for**
**return** $o$

**Limits of ownership.** Ownership is not guaranteed to be connected, perfectly balanced or a minimal cut, which are the balance and edge-cut objectives of a multilevel partitioner [36]. We measure the balance of owned training seeds across ranks and do not enforce it ($\text{CV}_w$ in Section 6.5). Vertices outside the active scope may keep no owner at all, since repair touches the training seeds only. A cache budget is not a strict upper bound when the mandatory low-degree base exceeds it (Section 3.7). Those last two non-guarantees were both observed in the sweep (Section 6.5). Training samples are not guaranteed to match unrestricted global sampling, because foreign neighbours outside the local view are invisible to the rank and are omitted. The evaluator is exact for two-layer mean-aggregating GraphSAGE and is not yet a general evaluator for arbitrary architectures,

and the native sampler supports exactly two fan-outs. Training accuracy is therefore not guaranteed to be unchanged, so we measure it and do not assume it.

### *3.3. Design Rationale*

We made each decision below as a deliberate trade. (i) **Ownership and cache are separate.** A vertex needs exactly one training owner, while availability needs no such exclusivity. (ii) **No on-demand remote fetch during training.** Fine-grained requests inside every sampled minibatch introduce request latency, synchronisation and unpredictable stalls, so SoftPart pays a one-off cache and materialisation cost and then runs a stable local hot path. A missing foreign neighbour is then a controlled approximation, and the runtime never fetches it. (iii) **Evaluation is different from training.** Approximate training access is part of the method under test, as neighbourhood sampling is already in every sampled GNN [32,37,38], whereas approximate evaluation would change the measurement itself. (iv) **The store stays global.** Ownership, cache contents, local views and the evaluation workload all derive from the same prepared graph, which is never split across workers.

### *3.4. Partitioner-Free Graph Store*

Input to the pipeline is a node-classification dataset with a fixed train/valid/test split. Preparation stores the topology in compressed sparse row (CSR) format, and features, labels and split indices as separate arrays, all mmap-able from shared storage, which is the CSR-plus-feature-tensor representation that DGL and DistDGL partitions also use [1,9]. We make every decision that changes the meaning of the graph (edge direction, symmetrisation, self-loops) here, and it becomes part of the dataset (Invariant 9), and no worker assignment exists once preparation has finished. Memory mapping gives every worker addressable access to the whole store without copying it into private memory, which is not the same as materialising the full feature matrix on every GPU (Section 6.7).

### *3.5. Online Locality Warm-Up*

Warm-up does not aim at a good cut. It observes which graph regions the actual training workload touches together. GNNLab's pre-sampling pass and PaGraph's static degree proxy collect this signal for cache placement [19,26], and here it decides ownership. We start from the training vertices because they generate the batches that update the model, and we shuffle their order deterministically and deal it round-robin to workers. Each worker runs a short exploration pass around its own seeds with the same fan-outs that GraphSAGE training will use [32], so the access pattern resembles the future workload. When a coverage target is configured, the warm-up order continues after the training seeds with the remaining vertices $V \setminus S$, so that discovery can go on if the training seeds alone do not reach the target. Warm-up checks coverage of the configured seed scope every 16 batches in the reference configuration and stops at the first checkpoint that reaches the target (98% of training seeds) or when the exploration pass ends (Algorithm 1). A batch cap exists in the configuration, and we disabled it in the reference sweep. After the loop ends, one more RESOLVE runs on the final claim sets, and the $o_{\text{pre}}$ that it produces is the map handed to repair, so that map reflects every claim made. Its cost is at most one sampled two-layer neighbourhood per seed, that is, $q_w\,(1 + k_2 + k_1k_2)$ vertex visits on rank $w$ for the $q_w \le |\texttt{seeds}_w|$ seeds processed before the pass stops, at most $|S|(1 + k_2 + k_1k_2)/W$ per rank plus the appended remainder, and fewer where a row's degree is below the fan-out, and the only communication is the claim exchange at the checkpoints. On OGBN-Products, most vertices outside the training seeds remain unclaimed (Section 6.5).

Algorithm A1 (Appendix A) gives the resolution rule in full, covering the claim gather, the hash over each vertex's own claimant list, and the $\perp$ assignment for the vertices that

no worker claimed. Per checkpoint, every rank receives $\sum_w |C_w|$ vertex ids and builds an $N$-entry owner map, and neither quantity falls with $W$.

### *3.6. Ownership Resolution and Repair*

Claims are exchanged at every coverage checkpoint, where RESOLVE (Algorithm A1) all-gathers the sorted claim sets and every worker builds the identical provisional owner map. This collective is an object all-gather of serialised Python 3.12 objects (ALLGATHEROBJECT of `torch.distributed` [33,34]), so its cost is not that of a tensor collective carrying the same number of ids, and the total number of claimed ids is logged while its byte volume is not. Checkpoints fall at fixed positions of each rank's own batch counter, and in the reported cells, every rank has the same number of warm-up batches, so all ranks enter each collective together. A sole claimant necessarily wins, and a collision is decided among the actual claimants only, by a deterministic hash of (vertex, seed), which preserves the warm-up's locality evidence. Identical claim sets, seed and world size yield the same pre-repair $o$, and the repair step adds a dependence on its traversal order, which the implementation fixes (Algorithm 2).

Training seeds that no worker claimed are repaired by Algorithm 2. Neighbour majority extends an existing ownership region wherever local evidence exists, and the hash fallback guarantees an assignment for vertices with no assigned neighbour, so that $o(v) \neq \perp$ for every $v \in V_{\text{act}}$ afterwards (Invariant 2). Hash placement is the classical default in graph processing and the METIS-free option in DistDGL and $P^3$ [1,4,39,40]. Our code supports further repair modes, and we used this one in every run reported here. A repaired vertex is placed with weaker or no locality evidence, and the fraction of seeds placed this way is reported in Section 6.5. This yields a *soft partition*, in which ownership is exclusive and availability is not. Exchanging all claim lists to all ranks, and building the $N$-entry owner map on every rank, are a known scalability limit, whose mechanism is decomposed in Section 3.10 and whose consequence is measured in Section 6.2.

Algorithm 2 costs one pass over the missing seeds with $\sum_{v \in \texttt{missing}} |\mathcal{N}(v)|$ neighbour reads and no communication, since every rank holds the full owner map. It writes $o(v)$ while iterating over `missing`, so its result depends on the traversal order. That order is the order of the stored split index lists, training seeds first, then validation and test seeds where in scope, each list as stored, which is ascending vertex id in the four prepared stores that we inspected. Those counts $c_w$ are read from the map being written, which already differs from the frozen pre-repair snapshot $o_{\text{pre}}$, so a vertex repaired earlier in the pass counts as a neighbour for a later one.

### *3.7. Memory-Budgeted Foreign-Vertex Cache*

A training vertex owned by one worker may have neighbours owned by others, the halo or ghost vertices of partitioned trainers [41,42]. Ignoring them would bias the sampler towards the ownership layout. SoftPart selects, before training, a bounded set of foreign vertices whose features and relevant topology are made available locally for the run, a memory-budgeted replication decision of the kind that GPU feature caches make [23,24,26]. The row budget is set relative to the expected number of foreign vertices under balanced ownership,

$$\rho = \text{round}\left(N \tfrac{W-1}{W}\right), \qquad b = \begin{cases} \text{round}(\rho f) & \text{if } b_{\max} = 0, \\ \min\left(\text{round}(\rho f),\ b_{\max}\right) & \text{otherwise,} \end{cases}$$

with $N$ vertices, $W$ workers, budget fraction $f$ and an optional absolute cap $b_{\max}$ (`cache_max_nodes`, where 0 means no cap). Our reference sweep uses $f = 0.25$ with the cap disabled (Section 5).

The reference policy is `degree_shortfall_random_fast`, implemented in native C++ and stated as Algorithm A2 (Appendix A). It works on the worker's owned training rows with the last configured fan-out $k$, and splits the cache into a mandatory base $M_w$ and an optional remainder, which we rank by usefulness to those rows. PaGraph ranks its cache by static out-degree [19]. The base exists because a row whose degree is at most $k$ is consumed whole by the model, so omitting one of its foreign neighbours removes information, and such neighbours are cached unconditionally. Shortfall scoring, in turn, counts per row the fan-out slots that owned neighbours cannot fill and spends them on foreign candidates, so a candidate useful to several rows accumulates points and is ranked first. We did not implement rotation policies in this runtime, so the cache stays static for the whole run.

Selection costs one scoring pass over the $|O_w \cap S|$ owned training rows, with $\bar{d}$ neighbour reads each for mean degree $\bar{d}$, plus one sort of the candidate set, and it is local to the rank. Scoring reads $\mathcal{N}(v)$ as the stored adjacency sequence and does not deduplicate it, so a duplicate adjacency entry, if a prepared graph contained one, would count with its multiplicity. Its mandatory base is added first, and the score-ranked remainder, sorted by descending score and then by ascending vertex id, fills whatever the row budget leaves, which makes the budget a soft cap. A rank whose base alone exceeds the budget overshoots it by exactly the excess of the base, and a rank whose candidate list is shorter than the remaining budget under-fills it. Our sweep contains both regimes (Section 6.5). The practical consequence is that host and GPU memory have to be sized from the mandatory base, since $f$ is only a soft cap. In the reference configuration, the base exceeds the $f = 0.25$ budget on three graphs (Section 6.5).

### *3.8. Local Training View*

A worker's available set is the union of its owned and cached vertices. Features of the available rows are materialised on the worker's GPU in the reference configuration, behind a global-to-local index map. That map is an `int64` CPU tensor kept at graph size, which buys one fast lookup per sampled vertex and does not shrink with the worker count. Local topology keeps exactly the edges with both endpoints available, and preserves graph-sized row addressing (an unavailable row is an empty row), so nothing is renumbered into a compacted partition. Any neighbour outside the view is invisible to training, as in the cluster- and subgraph-restricted samplers whose accuracy cost has been studied separately [29,43,44].

### *3.9. Sampling and DDP Training*

Every rank trains only its owned seeds, but every backward pass participates in a DDP collective [34], so all ranks must execute the same number of optimiser steps per epoch. That number, $\sigma = \max_w \lceil |O_w \cap S| / B \rceil$, is set by the rank with the most owned training seeds (Invariant 6), and a rank that exhausts its seeds reshuffles and wraps around, which preserves collective compatibility at the price that some seeds repeat within an epoch on smaller ranks. Our native two-layer sampler builds blocks from the output seeds backwards with an ordered-prefix property, so that each layer's first num_dst source rows are the destinations' own representations. Its unique- and position-building steps run in native kernels, so no per-batch Python dictionary sits in the measured sampler. Our model is two-layer mean-aggregating GraphSAGE, $h_{\text{dst}} = \Theta_{\text{self}} h_{\text{self}} + \Theta_{\text{neigh}} \,\text{mean}(h_{\text{neigh}}) + b$, with the bias $b$ on the self path only, since the neighbour projection is bias-free (Equations (1) and (2)), projecting neighbour rows before aggregation when the input width exceeds the output width (Section 4.2). A prefetch thread overlaps CPU sampling of future batches with GPU work, and its wait time is measured separately. In the training hot loop, there is no remote graph access, and the only per-step cross-worker communication is the DDP gradient

all-reduce. Exact evaluation adds one $H_1$ all-gather per validation plus a scalar all-reduce of the correct and target counts, outside the loop.

### *3.10. Cost Model*

This subsection collects the cost of each stage, in the notation of Section 3.1, and names the term that governs its behaviour with world size. Collective costs are counted as data volumes, without the latency and bandwidth constants of the standard $\alpha$–$\beta$ analyses [45,46]. Feature values are 4-byte floats, the CSR column indices are 4-byte and the row offsets 8-byte integers, and the owner map is a 2-byte integer per vertex. What follows is a decomposition, and the measurements of Section 6 supply its constants, so it is not a fitted model.

**Placement phase.** Warm-up sampling is local and communicates only at the coverage checkpoints, where the claim exchange of RESOLVE (Algorithm A1) is the term that scales badly. Every rank receives every other rank's claim list and materialises an $N$-entry owner map, so the exchanged volume is the number of checkpoints times the total number of claimed ids, and it does not fall with $W$. The work performed is bounded above by $|S|$ seeds plus the appended remainder, and it can be smaller, because the pass stops at the first checkpoint whose resolved map reaches the coverage target, and on OGBN-Products, the first checkpoint already ended the pass (Section 6.5). Claim volume tends to grow with $W$, since a vertex touched from two ranks' shares is claimed twice. That is the observed direction, and Algorithm 1 does not guarantee it. Ownership time is therefore the sum of the warm-up sampling, which is parallel and falls with $W$, this exchange, which does not, and the owner-map build, whose size is the graph. The claims that a rank receives are $W$ times the claims that it sends, so the exchange grows as $W^2$ times the per-rank claim volume while the map grows with the graph. The measured sizes are in Section 6.2. Cache selection is local, one scoring pass plus one sort (Section 3.7), the local-view build copies $|A_w| \cdot F \cdot 4\,\mathrm{B}$ to the device, and both shrink with $W$ because $|O_w|$ tends to $N/W$ (Section 6.7).

**Training, per optimiser step.** Sampling and feature gather read only the rank's local view and the materialised store, so they are local. Only the DDP all-reduce of the gradient crosses workers, and its volume is set by the parameter count alone, independent of $N$ and $M$. An epoch is $\sigma$ steps, with balanced owned seeds $\sigma \approx |S|/(BW)$, so training time falls roughly as $1/W$ until the fixed per-step cost stops being negligible. That happens on OGBN-Arxiv, whose epoch-time ratio against the baseline is flat over the three world sizes (Table 3).

**Exact evaluation, per validation.** One all-gather of $H_1$ delivers $N \cdot D \cdot 4\,\mathrm{B}$ to every rank, against $N \cdot F \cdot 4\,\mathrm{B}$ if input features were exchanged, a factor $F/D$ (6.25 to 64 on the reference graphs). Layer 1 is spread over the ranks by edge-balanced row sharding, and layer 2 runs on the target rows only. Sharding leaves two costs in place. Every rank projects the full feature matrix before aggregating its own row range, a term that does not shrink with $W$, and our reference configuration evaluates that projection on the host in single precision, off the GPU. The collective is padded to the largest shard, so it delivers $W \max_w |R_w| \cdot D \cdot 4\,\mathrm{B}$, and during the collective, the gathered buffer and the assembled $H_1$ coexist, so the transient footprint is about twice the resident $ND$ floats. Our reference protocol performs six validations and one test per run.

**Memory per rank.** Per rank, the GPU term is $M_{\mathrm{GPU}}(w) \approx (|O_w| + |K_w|) \cdot F \cdot 4\,\mathrm{B}$ plus the model, optimiser and sampled-block buffers, the term that motivates shared multi-GPU stores [25] and out-of-core designs where it does not fit [17,20]. Its CPU term does not fall with $W$, because the owner map is an $N$-entry `int16` array, the global-to-local map is an $N$-entry `int64` array, and the CSR is mmap-resident (Section 6.7).

**Table 3.** Paired comparison per cell. Accuracies are mean(sd) of five runs per side, sd in last-digit units. ‡ marks the two cells with one accuracy-anomalous run, kept in the mean and sd. † marks a difference significant after Holm correction (Welch test without the flagged runs; the first four graphs at 4–16 nodes form one family, IGB-hom-medium its own and the four 32-node rows a third). E2E is scheduler-reported wall-clock time (DistDGL without METIS partitioning). The geometric ratio is the ratio of geometric means with a 95% Welch interval on log E2E; the ratio including partitioning adds DistDGL's in-process partitioning time. The epoch-time ratio divides mean epoch times of the slowest rank. Ratios below 1 favour SoftPart GNN.

| Dataset/Nodes | SoftPart Test Acc. | DistDGL Test Acc. | Δ acc. | SoftPart E2E [s] | DistDGL E2E [s] | E2E Ratio (Mean) | E2E Ratio, Geometric [95% CI] | E2E Ratio Incl. Partitioning | Epoch-Time Ratio |
|---|---|---|---|---|---|---|---|---|---|
| OGBN-Arxiv (bidir.), 4 | 0.6382(31) | 0.6237(349) ‡ | +0.0145 | 20.0 | 80.0 | 0.25 | 0.27 [0.16, 0.47] | 0.23 | 0.28 |
| OGBN-Arxiv (bidir.), 8 | 0.6301(108) | 0.6284(45) | +0.0017 | 18.8 | 56.2 | 0.33 | 0.34 [0.26, 0.46] | 0.26 | 0.30 |
| OGBN-Arxiv (bidir.), 16 | 0.6086(109) | 0.6081(50) | +0.0005 | 19.2 | 64.6 | 0.30 | 0.32 [0.19, 0.56] | 0.23 | 0.30 |
| Reddit, 4 | 0.9497(8) | 0.9509(9) | −0.0012 | 81.6 | 179.0 | 0.46 | 0.45 [0.36, 0.57] | 0.34 | 0.34 |
| Reddit, 8 | 0.9482(6) | 0.9504(8) | −0.0022 † | 46.0 | 137.6 | 0.33 | 0.34 [0.26, 0.44] | 0.23 | 0.32 |
| Reddit, 16 | 0.9452(7) | 0.9478(6) | −0.0027 † | 36.6 | 95.0 | 0.39 | 0.39 [0.31, 0.49] | 0.23 | 0.30 |
| OGBN-Products, 4 | 0.7018(9) | 0.7016(60) | +0.0003 | 87.6 | 168.4 | 0.52 | 0.53 [0.43, 0.65] | 0.35 | 0.71 |
| OGBN-Products, 8 | 0.7004(17) | 0.6964(25) | +0.0039 | 59.0 | 119.8 | 0.49 | 0.50 [0.41, 0.60] | 0.30 | 0.59 |
| OGBN-Products, 16 | 0.6944(27) | 0.6889(25) | +0.0055 | 49.6 | 80.8 | 0.61 | 0.62 [0.50, 0.76] | 0.31 | 0.45 |
| IGB-hom-small, 4 | 0.7416(7) | 0.7347(161) ‡ | +0.0070 | 132.4 | 603.2 | 0.22 | 0.22 [0.18, 0.27] | 0.21 | 0.13 |
| IGB-hom-small, 8 | 0.7411(9) | 0.7416(12) | −0.0005 | 75.0 | 337.4 | 0.22 | 0.22 [0.18, 0.27] | 0.21 | 0.11 |
| IGB-hom-small, 16 | 0.7385(5) | 0.7405(4) | −0.0020 † | 63.6 | 212.8 | 0.30 | 0.28 [0.18, 0.44] | 0.26 | 0.09 |
| IGB-hom-medium [a], 4 | 0.7322(3) | 0.7325(4) | −0.0003 | 2238.6 | 4859.2 | 0.46 | 0.46 [0.45, 0.48] | 0.44 | 0.14 |
| IGB-hom-medium [a], 8 | 0.7321(3) | 0.7324(5) | −0.0002 | 1304.0 | 2790.8 | 0.47 | 0.47 [0.45, 0.48] | 0.43 | 0.12 |
| IGB-hom-medium [a], 16 | 0.7320(4) | 0.7328(3) | −0.0008 † | 1020.6 | 1733.4 | 0.59 | 0.59 [0.53, 0.65] | 0.52 | 0.10 |
| OGBN-Arxiv (bidir.), 32 [b] | 0.5498(315) | 0.5497(119) | 0.0000 | 24.4 | 50.8 | 0.48 | 0.47 [0.32, 0.68] | 0.44 | 0.37 |
| Reddit, 32 [b] | 0.9373(17) | 0.9414(20) | −0.0042 † | 41.8 | 82.2 | 0.51 | 0.50 [0.37, 0.67] | 0.30 | 0.29 |
| OGBN-Products, 32 [b] | 0.6819(38) | 0.6795(53) | +0.0024 | 65.4 | 71.8 | 0.91 | 0.91 [0.68, 1.20] | 0.37 | 0.39 |
| IGB-hom-small, 32 [b] | 0.7352(9) | 0.7376(4) | −0.0024 † | 61.8 | 151.0 | 0.41 | 0.40 [0.27, 0.57] | 0.31 | 0.11 |

[a] IGB-hom-medium: the SoftPart 4-node cell and all DistDGL cells ran under GPU driver 610.43.02 with the reference sweeps; the SoftPart 8- and 16-node cells ran later under driver 615.71.09 (Section 5). [b] Thirty-two nodes, locality arm against DistDGL in the reference configuration, five runs per side, driver 615.71.09 on both sides; the two systems ran in different waves. The partitioning time at 32 nodes is that of a 32-part partition, and the two ratios are ratios of means. With the modulo arm, the 32-node ratios are 0.38, 0.38, 0.44 and 0.25 (OGBN-Arxiv, Reddit, OGBN-Products, IGB-hom-small; Table 4); IGB-hom-medium has no DistDGL run at 32 nodes.

**Table 4.** Paired ownership control, locality warm-up against modulo ownership, five pairs per cell; a pair shares the seed and everything else but the ownership map. On IGB-hom-medium at eight and 16 nodes, the modulo arm requested twice the host memory (Section 6.7). Δ test is locality minus modulo accuracy (mean of per-pair differences, 95% paired *t* interval). Epoch-time and E2E ratios are modulo over locality, so values above 1 favour locality; geometric means of per-pair ratios with 95% paired *t* intervals on the log ratios. The steady-state ratio excludes the first epoch. Ownership times are pair means. The last five rows are at 32 nodes.

| Dataset/Nodes | Δ Test (Loc. − Mod.) | Epoch Ratio (Mod./Loc.) | Steady-State Epoch Ratio | E2E Ratio (Mod./Loc.) | Ownership Loc. [s] | Ownership Mod. [s] |
|---|---|---|---|---|---|---|
| OGBN-Arxiv (bidir.), 4 | +0.0014 [−0.0106, +0.0134] | 1.09 [0.97, 1.23] | 1.00 | 1.21 [0.74, 1.97] | 1.95 | 0.014 |
| OGBN-Arxiv (bidir.), 8 | +0.0016 [−0.0101, +0.0133] | 1.02 [0.74, 1.40] | 1.00 | 0.91 [0.44, 1.87] | 1.66 | 0.009 |
| OGBN-Arxiv (bidir.), 16 | +0.0002 [−0.0055, +0.0059] | 1.18 [0.92, 1.52] | 1.00 | 1.14 [0.58, 2.25] | 2.15 | 0.007 |
| Reddit, 4 | −0.0005 [−0.0016, +0.0007] | 1.08 [1.03, 1.13] | 1.07 | 0.87 [0.77, 0.98] | 13.33 | 0.014 |
| Reddit, 8 [a] | −0.0003 [−0.0015, +0.0010] | 1.14 [1.08, 1.20] | 1.08 | 1.10 [0.70, 1.74] | 8.33 | 0.010 |
| Reddit, 16 | 0.0000 [−0.0005, +0.0005] | 1.11 [1.00, 1.24] | 1.08 | 0.88 [0.63, 1.24] | 8.22 | 0.012 |
| OGBN-Products, 4 | −0.0016 [−0.0041, +0.0008] | 1.13 [1.10, 1.16] | 1.11 | 0.96 [0.80, 1.14] | 10.75 | 0.124 |
| OGBN-Products, 8 | +0.0003 [−0.0022, +0.0029] | 1.16 [1.11, 1.20] | 1.11 | 1.00 [0.76, 1.31] | 14.24 | 0.070 |
| OGBN-Products, 16 | −0.0012 [−0.0029, +0.0005] | 1.23 [1.13, 1.33] | 1.17 | 0.87 [0.61, 1.24] | 19.15 | 0.044 |
| IGB-hom-small, 4 | −0.0001 [−0.0012, +0.0011] | 1.47 [1.45, 1.49] | 1.40 | 1.05 [0.96, 1.15] | 22.14 | 0.054 |
| IGB-hom-small, 8 | +0.0004 [−0.0003, +0.0012] | 1.53 [1.50, 1.57] | 1.44 | 1.12 [0.84, 1.49] | 17.61 | 0.033 |
| IGB-hom-small, 16 | +0.0003 [−0.0004, +0.0009] | 1.14 [1.08, 1.21] | 1.06 | 1.12 [0.71, 1.75] | 15.06 | 0.023 |
| IGB-hom-medium, 4 | +0.0002 [−0.0005, +0.0008] | 1.47 [1.42, 1.53] | 1.39 | 0.53 [0.50, 0.57] | 1200 | 0.484 |
| IGB-hom-medium, 8 | −0.0002 [−0.0007, +0.0002] | 1.56 [1.48, 1.65] | 1.46 | 0.48 [0.45, 0.51] | 744 | 0.259 |
| IGB-hom-medium, 16 | −0.0001 [−0.0006, +0.0004] | 1.53 [1.51, 1.55] | 1.45 | 0.38 [0.27, 0.53] | 547 | 0.152 |
| OGBN-Arxiv (bidir.), 32 | −0.0003 [−0.0080, +0.0074] | 0.92 [0.79, 1.07] | 0.95 | 0.81 [0.56, 1.16] | 2.05 | 0.006 |
| Reddit, 32 | +0.0001 [−0.0005, +0.0007] | 1.13 [1.04, 1.22] | 1.15 | 0.75 [0.61, 0.93] | 7.68 | 0.008 |
| OGBN-Products, 32 | −0.0009 [−0.0016, −0.0002] | 1.21 [1.10, 1.33] | 1.21 | 0.49 [0.32, 0.73] | 32.06 | 0.030 |
| IGB-hom-small, 32 | +0.0002 [−0.0002, +0.0006] | 0.98 [0.83, 1.17] | 1.02 | 0.64 [0.32, 1.26] | 14.77 | 0.016 |
| IGB-hom-medium, 32 | 0.0000 [−0.0002, +0.0002] | 0.98 [0.96, 1.00] | 0.98 | 0.34 [0.25, 0.46] | 422 | 0.104 |

[a] Time statistics use the pairs with a valid time on both arms (Reddit, 8: 5 epoch pairs, 4 E2E pairs); accuracy uses all five pairs. In ten of the 60 pairs of the first four graphs at 4–16 nodes, all at 16 nodes, the modulo run ran under the later GPU driver; the 50 same-driver pairs give an epoch ratio (mod./loc.) of 1.19 against 1.18 over those 60 pairs.

**End-to-end.** $T(W) = T_{\text{own}}(W) + T_{\text{cache}}(W) + T_{\text{train}}(W) + T_{\text{eval}} + T_{\text{fixed}}$, with $T_{\text{train}} \propto 1/W$, $T_{\text{cache}}$ decreasing and then flat, $T_{\text{own}}$ falling more slowly than $T_{\text{train}}$ and growing on OGBN-Products, and $T_{\text{eval}}$, $T_{\text{fixed}}$ roughly constant. Only $T_{\text{train}}$ carries the $1/W$, so the placement and fixed terms that a per-epoch ratio excludes are what move the end-to-end ratio the other way (Section 6.1).

Only the gradient all-reduce thus crosses workers during training, at the price of a claim exchange that does not fall with $W$.

## 4. Exact Distributed Evaluation

Models trained under different layouts are comparable only if one function of the graph, features and parameters scores them. For two-layer mean GraphSAGE, that function is computed exactly across workers with one $D$-wide all-gather.

### *4.1. Why the Training View Cannot Be the Measurement*

Accuracy measured on a locality-restricted view would mix model quality with ownership and cache coverage, which boundary-approximation methods separate by measuring against full neighbourhoods [29,30]. Validation and the final test therefore run on the global prepared graph with every neighbour of every target vertex, a *full-neighbour* evaluation whose result does not depend on the training layout (Section 6.8).

### *4.2. The $H_1$ Decomposition*

For two-layer mean-aggregating GraphSAGE with normalised adjacency $\bar{A}$ (row $i$ holds $1/\deg(i)$ on $i$'s neighbours), features $X \in \mathbb{R}^{N \times F}$ and hidden width $D$,

$$H_1 = \text{ReLU}\big(X\Theta_{\text{self},1} + \bar{A}\,(X\Theta_{\text{neigh},1}) + b_1\big), \tag{1}$$

$$Z = H_1\Theta_{\text{self},2} + \bar{A}\,(H_1\Theta_{\text{neigh},2}) + b_2, \tag{2}$$

with $\Theta_{\text{self},2}, \Theta_{\text{neigh},2} \in \mathbb{R}^{D \times C}$, since layer 2 maps the hidden width to the $C$ classes. Row $i$ of $\bar{A}$ is the zero row for $\deg(i) = 0$, in the evaluator and in the training sampler alike. Bias terms $b_\ell$ sit on the self path only, and our model's neighbour projection carries no bias (Section 3.9). Under that assumption, and because mean aggregation is linear, $\bar{A}(X\Theta) = (\bar{A}X)\Theta$, so projecting before communicating reduces the exchanged width from $F$ to $D$ without changing a single value. That particular ordering is the `lin_before_mp` option of DGL's `SAGEConv` [9], so the identity itself is not new. What we add is the protocol built on it, and in particular its independence from the training layout.

**Scope of the identity.** The identity needs the neighbour term to be linear in the neighbour features. It holds for mean, sum and GCN-normalised aggregation. It fails for max aggregation and for attention, whose weights depend on the features. For $L$ layers of the same form, one all-gather of $H_\ell$ ($N \cdot D_\ell \cdot 4\,\text{B}$) follows each of the first $L-1$ layers. The implementation realises $L = 2$ only, because the native sampler takes exactly two fan-outs.

**Assumptions.** The protocol assumes that every rank can read the prepared $G$ and $X$ in full and that the forward pass runs in evaluation mode with dropout disabled. It also assumes that the neighbour transform of each layer is linear without bias, as in our model. The bias-free form is what lets the $D$-wide exchange stand in for the $F$-wide one, while the row-selection argument below does not depend on it. The evaluator is then *layout-independent*, since the inputs of the distributed $H_1$ computation and of the target-subset evaluation are $G$, $X$, the model parameters $\theta$ and the target set $S_{\text{val}} \cup S_{\text{test}}$ only, and no placement object enters them. Two models obtained under any two training layouts are therefore scored by the same function, which is what makes the cross-system comparison of Section 6.3 well posed. The sharded computation is exact as well, because

row selection commutes with both matrix products and with the element-wise ReLU, $(\bar{A}\,Y)[R_w] = \bar{A}[R_w, :]\,Y$ for $Y = X\Theta_{\text{neigh},1}$. Shards of $H_1$ computed from Equation (1) and concatenated in row order therefore equal the single-machine $H_1$ row for row. Applied to Equation (2), with $H_1$ in place of $X$, the same argument shows that the logits computed on any split of the target rows equal the single-machine logits. The single-machine reference of Section 6.8 may therefore aggregate before projecting as well as after, because the two orderings agree by the identity above.

**Exactness and numerical reproducibility.** Exactness is a mathematical statement. In exact arithmetic, the sharded evaluator computes the same function as a single-machine GraphSAGE of the same weights. Numerical reproducibility is a measured property. In floating-point arithmetic, the agreement depends on the matmul precision, the backend, the chunking of Section 4.3 and the summation order [47]. Because accuracy is an arg max over logits, a small logit deviation can still change the class of a target whose top two logits lie within it. We therefore report the maximum deviation and the number of changed predictions (Section 6.8).

### *4.3. Distributed Protocol*

Rows of $\bar{A}$ are sharded across ranks, in the reference configuration by approximately equal edge count to avoid stragglers from unequal degrees. Each rank computes its row range of $H_1$ (Equation (1)) in chunks that bound the intermediate sparse tensors, and the chunking bounds memory only; it does not sample. Shards are exchanged in a single padded `all_gather`, after which padding is removed and global row order restored. Every rank then holds the full $H_1$ ($N \times D$ floats). We chose that trade deliberately, memory for one bulk collective and a fast second layer, and it spares the exchange of the $F$-wide feature matrix. Target vertices are split into contiguous, balanced slices, one per rank, and the second layer (Equation (2)) is evaluated per target chunk. Accuracy is the ratio of globally summed correct and target counts (Invariant 7), and a mean of per-rank accuracies is never formed. The $H_1$ computed by the last validation after the final optimiser step is reused for the final test when its version key matches (Invariant 8), and that reuse changes the cost of the test, while its predictions stay the same.

**Scope of the verification.** We checked the implementation against a single-machine full-neighbour reference at four ranks on four graphs (OGBN-Arxiv, Reddit, OGBN-Products, IGB-hom-small), on SoftPart checkpoints and on 20 DistDGL-trained checkpoints (four graphs, five seeds). Two further cases are OGBN-Arxiv at $D = 256$, a case with $D > F$, and a sum-aggregating model on three graphs. Section 6.8 reports the outcome. The $k$-layer protocol is analytic only. The width reduction helps only where $F > D$, which holds for the four graphs ($F \in \{100, 128, 602, 1024\}$ against $D = 16$), and DistDGL was not run at $D > 16$.

### *4.4. Relation to Layer-Wise Inference, Full-Graph Training and Distributed Inference*

Computing $H_1$ for all vertices with full neighbourhoods and then $Z$ from $H_1$ is the layer-wise inference pattern of the reference GraphSAGE implementations [9,32], and DistDGL ships a distributed variant over the partitioned graph. The evaluator described here differs from it. It runs on the global prepared store, so its result is independent of the training layout. It replicates the narrow $H_1$ with one all-gather, so no intermediate row is pulled through a key–value store per batch. It is the measurement instrument of a cross-system comparison, and it serves no single system as an inference path. Full-graph training systems evaluate on complete neighbourhoods, so exactness at evaluation time is not a separate instrument for them [14,48], and several approximate at training time instead [30,41,42]. Distributed full-graph inference systems [49] optimise throughput for

every vertex, and no system that we found provides an evaluator for a sampled, locality-restricted trainer.

## 5. Experimental Setup

Both systems share the container, GPU partition, prepared graphs, model and hyperparameters, so only the runtime and its configuration differ.

### *5.1. Hardware and Software*

All training runs execute on the `gpu` partition of the Komondor supercomputer (KIFÜ, Debrecen, Hungary) [50]. Its 58 nodes are HPE Cray EX235n blades (Hewlett Packard Enterprise, Spring, TX, USA), each with one 64-core AMD EPYC 7763 processor, 256 GB of memory and four NVIDIA A100-SXM4-40 GB GPUs, and two 200 Gb/s HPE Slingshot links per node connect them in a Dragonfly topology. Each run starts one trainer process per node with Slurm [51] `srun --mpi=none`, and every process gets one GPU (`--gres=gpu:1`) and 25 CPU cores. In the runs of Table 3, the host-memory request is 8 GiB for every SoftPart run on the four smaller graphs, 32, 24 and 16 GiB on IGB-hom-medium at 4, 8 and 16 nodes, and 8–32 GiB per cell on the baseline side, largest on OGBN-Products. The nodes were not requested exclusively, so the remaining cores, memory and GPUs of a node could be scheduled to other jobs during a run. The prepared graph store, the code and all run outputs reside on the `/scratch` Lustre tier. SoftPart ranks communicate through PyTorch Distributed [33] with the Gloo backend [52]. Every run uses one GPU per node, and we measured no multi-GPU node. The container is `nvcr.io/nvidia/dgl:25.08-py3` (Ubuntu 24.04, Python 3.12.3, PyTorch 2.8.0a0, CUDA 13.0 with cuDNN 9.12, NCCL 2.27.6, DGL 2.5), used by both systems. The reference sweeps ran under GPU driver 610.43.02 and the later runs under 615.71.09. The image digest and the container definitions are in the repository, and the image runs under Singularity [53] on the cluster. The image is extended with an SSH server that serves only the `ip_config` launcher of DistDGL [1,9], and the SoftPart runtime imports neither DGL nor the SSH layer. The baseline runs DGL's reference distributed GraphSAGE example on the `dgl.distributed` path, without the GraphBolt sampling backend, with an evaluation batch of 5000 targets and validation every five epochs. Its METIS partitioning step, measured and reported separately, is a single-node `cpu`-partition job with 25 cores and a per-dataset memory request.

### *5.2. Datasets*

Five node-classification graphs are used (Table 2), OGBN-Arxiv with symmetrised edges (*bidir*), Reddit, OGBN-Products, IGB-hom-small and IGB-hom-medium [32,54,55]. IGB-hom-medium, with 10 M vertices, is the largest graph. Each graph is converted once into the prepared store of Section 3, and the DistDGL partitions are built from the same prepared graph, so Invariant 9 holds. No training job was accepted before the global vertex and edge counts of the DistDGL graph matched the prepared store, and they match on all five graphs (Table 2). A further check of the same graph on both sides is described in the paragraph on reproducibility below.

### *5.3. Protocol*

Both systems train the same model with the same hyperparameters, a two-layer mean GraphSAGE with hidden width 16 and fan-outs (10, 25). Training uses batch size 512 per rank, Adam with learning rate 0.003 [56] and dropout 0.5 for 30 epochs, with validation every five epochs and the final test on the epoch-30 model. The baseline splits training vertices evenly across trainers with DGL's `node_split` (`force_even`), and SoftPart wraps smaller rank shards so that every rank executes the same number of steps (Invariant 6). We ran five runs per cell at 4, 8 and 16 nodes with one GPU each, 60 runs per system

over twelve cells, and every reported run completed. IGB-hom-medium was run at the same world sizes. We also ran 32 nodes on all five graphs for SoftPart and on the four reference-sweep graphs for DistDGL. The SoftPart seeds are 2026082601 to 2026082605, and each DistDGL run uses its own partition-relabelling seed. No valid run was excluded on speed or accuracy grounds, and one semantically invalid DistDGL run was replaced under an explicit criterion (Section 7). We set $f = 0.25$ and disable the cap $b_{\max}$. DistDGL starts from a pre-built METIS partition whose time we report separately (Sections 6.1 and 7). It is excluded from the primary end-to-end time, so the primary ratios are conservative in DistDGL's favour.

**Baseline configuration.** DistDGL runs in one fixed configuration of two sampler workers, one trainer and one server per node [1,57], applied unchanged to every cell without per-cell tuning. It samples on the `dgl.distributed` path without GraphBolt and builds a fresh METIS partition per run from a seeded vertex relabelling. To bound the effect of this choice, we measured six further sampler, server and batch-size configurations on OGBN-Products and Reddit at 8 nodes, five seeds each, with the reference configuration rerun later in SoftPart's reference wave (Table S4). The tuned runs are from an earlier wave, and SoftPart kept batch size 512. A second set of five runs of the twelve DistDGL cells saved a checkpoint in every run. It ran in earlier waves and under an earlier launch setting, and at 8 nodes it adds 12–24 s of fixed time per run to the same-wave reference, so it is no timing reference.

**Statistics.** End-to-end times are ratios of geometric means with 95% Welch intervals on log E2E. Accuracy differences (SoftPart minus DistDGL) carry 95% Welch intervals, using the four unflagged runs in the two flagged cells. The Holm correction applies within each family: Table 3 (12 cells), IGB-hom-medium (3), fan-out (16), tuned baseline (6) and, at 32 nodes, each ownership arm against DistDGL (4) and the ownership control (5). Non-inferiority is one-sided at a margin of 0.006 (lower 90% bound above $-0.006$). The margin is the 0.006 band of the submitted version, so it is post hoc for the twelve reference cells and fixed in advance for the cells run after submission. The ownership control uses pairs matched by seed index and a two-sided Wilcoxon signed-rank test on the per-pair ratios.

**Memory monitor.** One job-scoped monitor measured both systems once per second on every node. It records the RSS, PSS and USS of the job's own processes, selected by control group, their GPU memory, the job's memory control group and the CUDA allocator peak. We report per-node peaks, with one worker per node, as the maximum over nodes and the mean over five seeds, 75 runs per system.

**Reproducibility.** The reference sweep used GPU driver 610.43.02 and the later runs 615.71.09 (Section 5.1). For one configuration, the drivers differ by $-3$ to $+8$% in epoch time and by $-19$ to $+28$% in end-to-end time, with different seeds and node conditions. Seeds, the matmul-precision switch and all run specifications are in the repository, with a guide that reproduces one cell of Table 3. Both systems start from the same prepared CSR and feature store, and we exclude this one-time preparation on both sides. On all five graphs, counts, the SHA-256 of the sorted degree sequence, self-loops, labels, splits and feature sums agree between one four-part DistDGL partition and the store.

**Two evaluators.** SoftPart GNN scores validation and test with the evaluator of Section 4, in-run every five epochs and once on the epoch-30 model. DistDGL scores with DGL's full-neighbour layer-wise inference on the same schedule and reports the mean of the per-rank accuracies, whereas SoftPart reports the ratio of the global counts (Invariant 7). The two aggregations differ by at most $W/(2\,|\text{targets}|)$, $3.4 \times 10^{-4}$ in the worst cell. The DistDGL runs of Table 3 saved no checkpoint, so each end-to-end time includes its own system's evaluation cost.

### *5.4. Metrics*

*E2E* is the Slurm-reported elapsed time of the job, including container and process start-up, as in studies that measure partitioning alongside training [8]. *Internal E2E* is the runtime interval of the slowest rank (0.2% from rank 0 in the median). We split the E2E time of both systems into four parts. For SoftPart, *placement* is the locality warm-up and resolution, cache selection and local-view build, *training* is 30 times the mean epoch time of the slowest rank, *evaluation* is the six periodic evaluations plus the final test, and *other* is start-up and teardown. DistDGL has no placement inside the run, and its partitioning is a separate job with its own column. Its evaluation is the six periodic evaluations, each computing validation and test. The *epoch time* is the mean over epochs of the slowest rank's epoch time and excludes evaluation (rank 0 differs from it by 0.16% in the median on the reference runs). The *steady-state epoch time* leaves out the first epoch and is quoted next to the primary figure. Memory is reported per node as anonymous plus shared pages, PSS and the CUDA allocator peak. Accuracy is the exact full-neighbour accuracy of Section 4.

## 6. Results

At four to 16 nodes, SoftPart GNN finishes in 0.22–0.62 of DistDGL's time on five graphs, and its accuracy stays within 0.006 of DistDGL's on four of them. This section shows what the placement costs, where the remaining deficits come from, and how far the numbers can be trusted.

### *6.1. End-to-End Time*

Over the twelve cells of the reference sweep, SoftPart GNN completes the full run in 0.22–0.62 of DistDGL's time as a ratio of geometric means (Figure 2, Table 3). The advantage is significant after correction in every cell (Table 3), and every SoftPart run is faster than every DistDGL run of its cell. The SoftPart time spans the whole run, from the warm-up to the final test.

DistDGL's primary figure excludes its METIS partitioning, a cost that comparative studies measure separately [7,8], as did our own earlier DistDGL case study [58]. Counting each run's own partitioning (load, METIS and save) lowers the ratio of means from 0.22–0.61 to 0.21–0.35, and the ratios are conservative because the container start of that job is not counted. Figure 2a draws each cell's partitioning time as a hatched extension of its DistDGL bar.

The ratio of mean epoch times of the slowest rank spans 0.09–0.71 (Table 3, Figure 2b), and its low end lies on IGB-hom-small, where SoftPart's own placement phase dilutes the end-to-end figure (Section 6.2). The end-to-end advantage is smallest on OGBN-Products at 16 nodes, where 66% of DistDGL's time and 75% of SoftPart's lie outside the 30 epochs, so there it rests partly on fixed costs that both systems pay. SoftPart's exact evaluation costs less than DistDGL's in ten of the twelve cells and more on OGBN-Arxiv at eight and 16 nodes, because its per-rank projection does not shrink with the worker count (Figure 2a, Section 3.10).

On IGB-hom-medium, with 10 M vertices, the ratio is 0.46–0.59 (0.43–0.52 with the partitioning) and every SoftPart run is faster. The three cells are non-inferior at 0.006, and the 16-node deficit of 0.0008 is significant (Table 3).

At 32 nodes on four graphs, the ratio is 0.40–0.91 with the locality ownership and 0.25–0.44 with the modulo ownership (Tables 3 and 4, last rows and footnote). On OGBN-Products, the locality arm's advantage falls short of significance, because its ownership stage takes 32 s. On Reddit, SoftPart's accuracy falls outside the 0.006 band, with a deficit of 0.0042, which is the largest of any cell, while on IGB-hom-small, it is lower by 0.0024,

significant and non-inferior at 0.006. These times include SoftPart's own placement, whose cost Section 6.2 isolates.

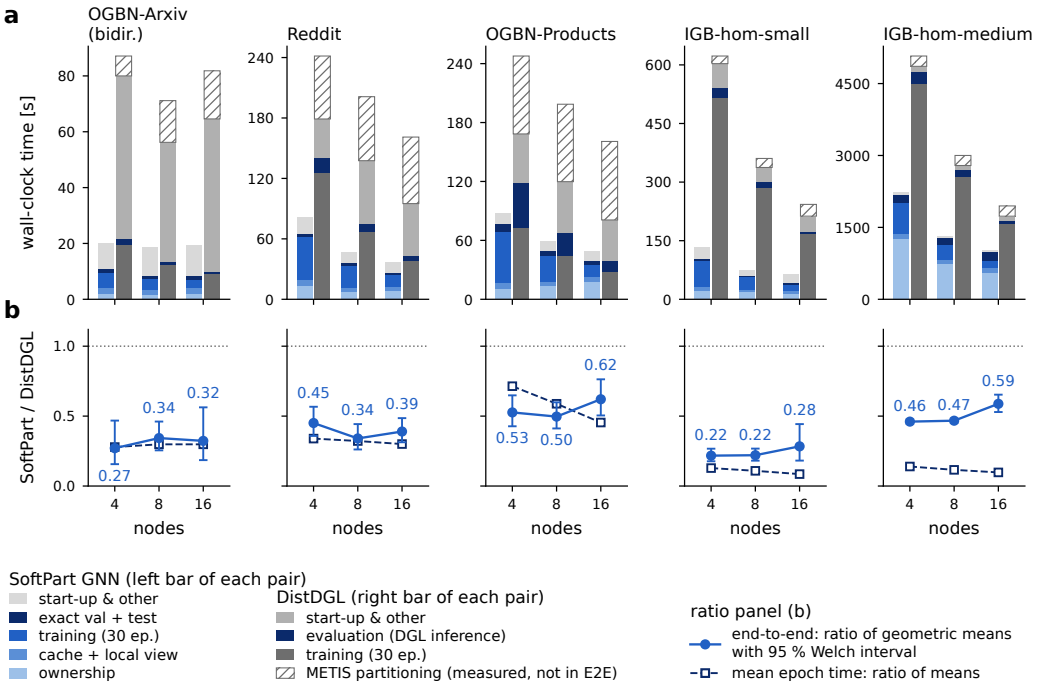



**Figure 2.** (**a**) Slurm-reported end-to-end time per cell, without DistDGL's partitioning, decomposed into the parts defined in Section 5. SoftPart GNN is shown in blue and DistDGL in grey. The hatched extension above each DistDGL bar is that cell's measured METIS partitioning time, which is not part of the plotted time. (**b**) Ratio SoftPart/DistDGL for the same end-to-end time (filled) and mean epoch time (open). Values below 1 favour SoftPart. The end-to-end ratio is the ratio of geometric means and carries a 95% Welch interval on the logarithm of the end-to-end time.

### *6.2. The Cost of Placement*

The placement that buys this speed takes 20.6–55.0% of SoftPart's internal end-to-end time on the twelve cells of the reference sweep and becomes the bottleneck at ten million vertices. Placement time falls more slowly than training time, or even grows, so its share rises with the number of workers on every dataset (Figure 3a). Ownership is the largest placement term in eleven of the twelve cells. On OGBN-Products, it grows from 10.9 to 18.5 s between four and 16 nodes while training falls from 52 to 12.5 s (Figure 3b). Holding it at its four-node value would give a ratio of 0.52 at 16 nodes, as at four nodes, so this growth accounts for the degradation of that graph's end-to-end ratio.

Timers on all five graphs at four, eight and 16 nodes (Figure 3c) show the owner-map build as the largest component of the ownership stage on OGBN-Products, IGB-hom-small and IGB-hom-medium, and the warm-up on Reddit. Only on OGBN-Products does the owner-map build grow with the world size, while the all-gather time grows by 44% and the warm-up stays nearly constant. On IGB-hom-medium, every component falls with $W$, yet the owner-map build is 78% of the ownership stage and the stage is 54–61% of the run, so at 10 M vertices, placement is the bottleneck. At 32 nodes, the stage is 49% of the end-to-end time on OGBN-Products and 60% on IGB-hom-medium.

The owner map takes 2 B per vertex, which is 20 MB on IGB-hom-medium. On OGBN-Products, a rank sends about 4.1 MB of claims at every world size and receives about $W$ times that. At a fixed number of exchanges, the all-gather traffic therefore grows as $W^2$ times the per-rank claim volume. On IGB-hom-medium, a rank receives up to 168, 275 and 405 MB per exchange at four, eight and 16 nodes, so the 16-node figure already exceeds

the 64-worker extrapolation below. The exchange runs 58, 30 and 16 times per run, and the total that a rank receives falls from 7.2 to 5.6 and 4.3 GB. The all-gather is a collective whose per-rank received volume does not fall as participants are added [46]. The closed form, which these runs do not test, gives a 200 MB owner map per rank at $10^8$ vertices, and at $W = 64$ with OGBN-Products-like claims, a rank would receive about 262 MB per exchange. The time of the owner-map build has no closed form in these data.

In the eight-node OGBN-Products cell, measured in one wave for the fan-out and tuning comparisons, placement takes 18.1 s of SoftPart's 57.4 s run, while DistDGL's 97.4 s leave out 80.3 s of partitioning in a separate job, longer than SoftPart's whole run. The corresponding cell of the reference sweep has 59.0 and 119.8 s, and Figure 2a decomposes every reference-sweep cell in the same way. The local view that this placement builds is restricted, so its cost in accuracy is measured next.

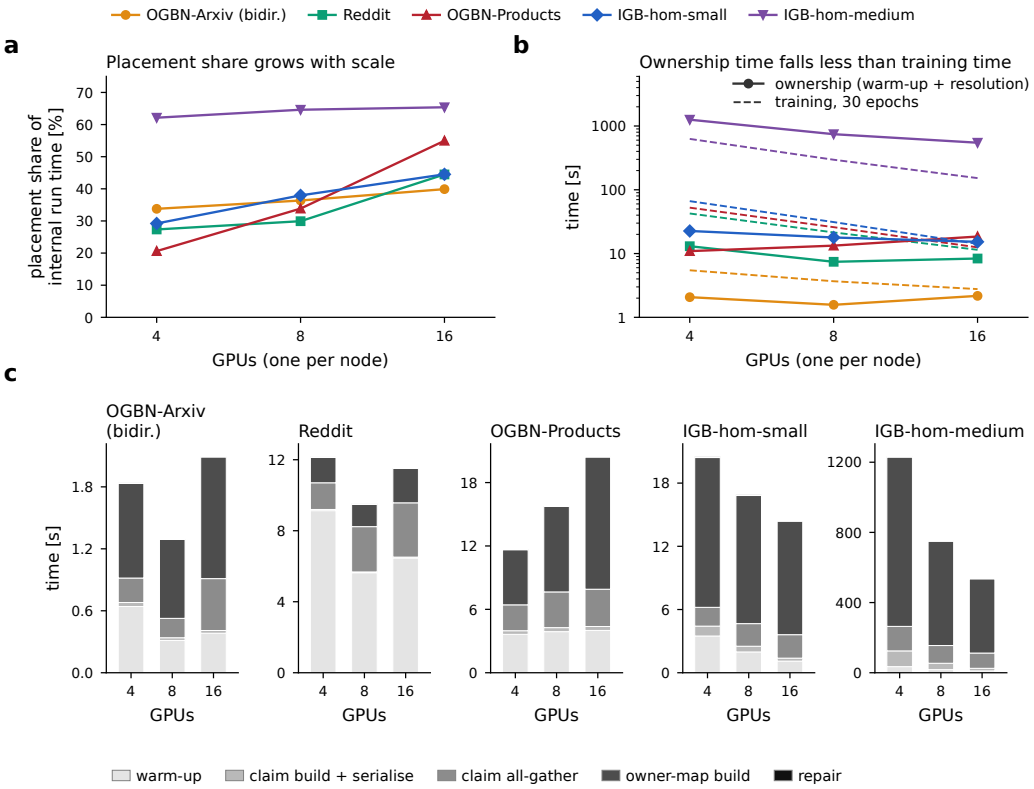



**Figure 3.** Placement diagnostics, means of 5 seeds. (**a**) Share of SoftPart's internal run time spent in the placement phase. (**b**) Ownership time (filled) against 30 epochs of training (dashed). (**c**) Seconds spent in the warm-up, the all-gather and the owner-map build of the ownership stage per graph and world size. Slowest rank of each run.

### *6.3. Model Quality*

Test accuracy stays within 0.006 of DistDGL's on four of the five graphs at four to 16 nodes, with OGBN-Arxiv left open and Reddit outside the band at 32 nodes (Figure 4, Table 3). The two largest differences in SoftPart's favour are on OGBN-Arxiv and IGB-hom-small at four nodes. Each traces to one flagged DistDGL run (Section 7) and shrinks to $-0.0003$ and $-0.0002$ without it. Without the flagged runs, SoftPart minus DistDGL ranges from $-0.0027$ to $+0.0055$ over the twelve cells.

After correction over the twelve cells, SoftPart is less accurate on Reddit at eight and 16 nodes and on IGB-hom-small at 16 nodes (Table 3). IGB-hom-medium at 16 nodes is a fourth (Section 6.1), and no deficit at 4–16 nodes exceeds 0.0027 (Table 3). The OGBN-Products differences in SoftPart's favour at eight and 16 nodes do not survive the correction,

so we do not read them as an advantage. SoftPart is non-inferior at the margin of 0.006 in every OGBN-Products, Reddit and IGB-hom-small cell. On OGBN-Arxiv, non-inferiority is not established, because the lower bounds of $-0.0147$, $-0.0087$ and $-0.0101$ at four, eight and 16 nodes leave five runs per side compatible with a deficit of up to about 0.015. Among the cells at 4–16 nodes, SoftPart's seed standard deviation is largest on OGBN-Arxiv at eight and 16 nodes, up to 0.0109 against at most 0.0060 for DistDGL outside the flagged cells.

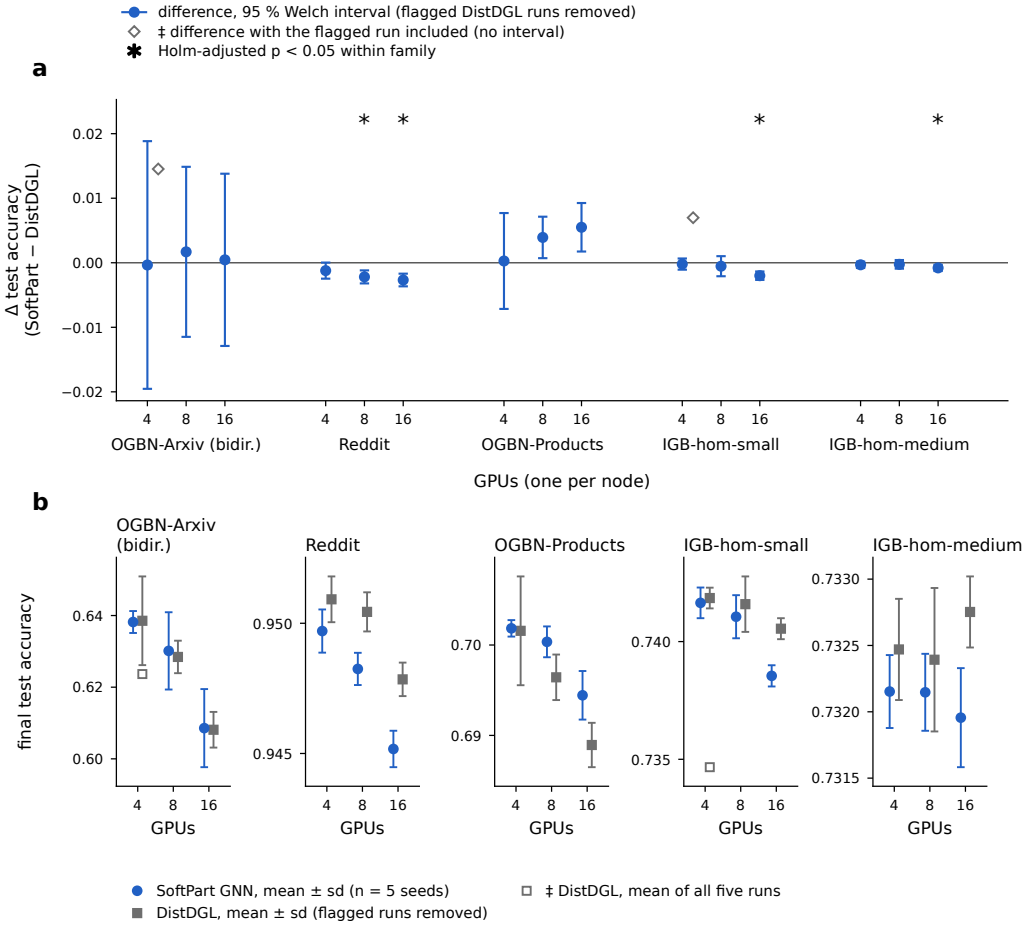



**Figure 4.** Final test accuracy of the epoch-30 model. (**a**) Difference SoftPart − DistDGL per cell, 5 seeds against 5 runs. The whiskers are the 95% Welch interval of that difference in means. Significance after the Holm correction is given in Table 3. (**b**) Absolute accuracy, mean ± sd on both sides. Cells marked ‡ contain one flagged DistDGL run each.

A second set of five runs of all twelve DistDGL cells, taken in the same week, shows neither anomaly. Table 3 without the flagged runs agrees with it within 0.0030 in eleven of the twelve cells, the two flagged cells included. On OGBN-Arxiv at 16 nodes, the second set is lower by 0.012, which is of the size of SoftPart's largest seed standard deviation.

Accuracy decreases with world size in both systems, from four to 16 nodes by 0.0296 for SoftPart against 0.0304 for DistDGL on OGBN-Arxiv without its flagged run, and by 0.0045 against 0.0031 on Reddit. At 32 nodes, OGBN-Arxiv falls further, to 0.550 for both systems, and the interval of the difference, $[-0.038, 0.038]$, is too wide to establish non-inferiority, so the comparison there is left open (Section 6.1, Table 3). We read the common trend as an effect of the shared training regime. With a fixed per-rank batch of 512, the global batch grows with the number of workers, so 30 epochs contain fewer optimiser steps [59,60]. No measurement here isolates this from the local view, and a 16-node run at per-rank batch 128 would separate the two. These conclusions hold for the measured

graphs, configurations and world sizes, and the next two subsections examine the local view as a candidate source of the remaining deficits.

### *6.4. Coverage of the Local View*

The restricted local view fills most of the sampled neighbour slots, and across cells, the share that it misses shows no detected relation to the accuracy difference to DistDGL, although within a graph the two grow together and cannot be separated. We sampled the same seed batches from the local view and from the full graph on all fifteen cells at four to 16 nodes (per-cell values in the repository). The local view cannot fill 2.4–21.0% of the sampled neighbour slots, a share that grows with the world size and is concentrated in the second expansion. The Kullback–Leibler divergence of the per-row fill counts is 0.003–0.28 on four graphs and close to zero on Reddit despite its missing slots, so it does not show which neighbours are lost. Across the cells, we detect no relation between the missing share and the accuracy difference to DistDGL. Within a graph, the missing share grows with the world size, so on IGB-hom-small and Reddit, whose deficit grows with both, the two effects cannot be separated. We therefore name the restricted local view as a candidate source of the Reddit and IGB-hom-small deficits, and we have not isolated it. Validation accuracy comes within 0.005 of its final value at most three epochs apart in the two systems, at less than one evaluation interval, so at this resolution the local view does not slow convergence.

### *6.5. Placement Diagnostics and Cache Budget*

The warm-up itself places nearly every training seed, the cache exceeds its nominal budget on three graphs, and the speed advantage holds at other fan-outs and against a tuned baseline. In all 60 runs of the reference sweep, the warm-up stopped at a coverage checkpoint, on OGBN-Products always at the first one, and repair (Section 3.6) touched at most 1.8% of the training seeds. Graph-wide assignment after repair (Section 3.1) is 98.3–99.9% on OGBN-Arxiv, Reddit and IGB-hom-small, while 21.3–40.3% of the OGBN-Products vertices have no owner, the most at four nodes. SoftPart is nevertheless non-inferior to DistDGL at the 0.006 margin in all three OGBN-Products cells (Section 6.3). Owned training seeds are balanced across ranks to within a coefficient of variation of 0.1–1.0%.

**Cache budget and composition.** Under the reference policy, the mandatory low-degree base exceeds the budget of $f = 0.25$ on three graphs and is kept in full (Sections 3.2 and 3.7). The overshoot reaches 2.54 times the budget on IGB-hom-small and 2.12 times on IGB-hom-medium at four nodes and falls to about the budget or below at 16 nodes (Section 3.7). Host and GPU memory therefore have to be sized from the mandatory base. At four nodes, with one seed per cell, $f = 0$, 0.10 and 0.25 cache the same rows on OGBN-Arxiv and IGB-hom-small, and $f$ binds only on Reddit. Accuracy moves by at most 0.0014 across these budgets, while running without a cache costs 2.3 percentage points on OGBN-Arxiv and 0.5 on IGB-hom-small and Reddit.

**Strict budget sweep.** A strict policy budgets $f$ times the owned rows, and we probed it at three low budgets only (no cache, 5% and 10%) at 16 nodes on five graphs with five seeds (Table S2). The strict 10% is 2–3% of the reference budget in rows, and the soft column comes from the separate reference sweep of Table 3. Accuracy rises with the budget on four graphs, most on OGBN-Arxiv. The soft budget costs up to 3.8 times the GPU memory of the strict 10% and 2.2–2.5 times the epoch time on OGBN-Products and Reddit. On Reddit, a small cache lowers the accuracy below that of no cache (0.921 against 0.933), and two simulations in the repository are consistent with this. Reddit's uncached view already fills the fan-out for the largest share of training rows among the four graphs, and a small degree-ranked cache supplies about ten times its share of the true neighbour slots.

The budget does not change communication, because the training step fetches no remote features at any budget.

**Fan-out sensitivity.** We ran both systems at four fan-outs from 5, 10 to 25, 50 on four graphs at eight nodes (Table S3). SoftPart is faster in every run of all 16 cells, with ratios of 0.17–0.79 that rise with the fan-out on OGBN-Products and fall on IGB-hom-small. After correction, only Reddit differs in accuracy, where SoftPart is lower by 0.0017–0.0041 at every fan-out. The 10, 25 point is not strictly comparable with the other three (Table S3, footnote a).

**Tuned baseline.** The six further DistDGL configurations of Section 5 ran on two graphs at eight nodes (Table S4). At batch size 512, they do not shorten the end-to-end time on OGBN-Products and gain 9% on Reddit, although their best epoch times are 8–11% lower. Batch size 2048 is 21% and 37% faster, at accuracy changes of $-0.048$ and $-0.010$. SoftPart is faster in every run of every comparison, at 0.44–0.59 of the fastest batch-512 configuration and 0.63–0.75 of batch size 2048, where it is also more accurate. The advantage narrows with tuning.

### *6.6. Paired Ownership Control*

Against modulo ownership with the same cache, the warm-up shortens the epoch on four of the five graphs at four to 16 nodes at the same accuracy, with the epoch saving absorbed by the placement time end to end. The paired control changes only the ownership map and keeps the seed and every other setting. On IGB-hom-medium at eight and 16 nodes, the modulo arm also requested twice the host memory (Section 6.7). One arm uses the locality warm-up of Section 3, the other $o(v) = v \bmod W$, a fixed exclusive assignment that needs no warm-up sampling, claim exchange or repair pass. We ran every pair at $f = 0.25$ (Section 6.5) with five seeds per arm, 15 pairs per graph at four to 16 nodes and five at 32 nodes (Table 4). The arms are matched by seed index but are independent runs. Ratios in this section are locality over modulo, so a value below one favours the locality arm, and the table gives the inverse.

The effect differs between graphs, so the per-graph results carry the conclusion. On four graphs, the locality arm has the shorter mean epoch in 59 of 60 pairs, by 7 to 36 percent per cell (Table 4). On OGBN-Arxiv, no steady-state difference is detected (ratio 1.00, $p = 0.72$). Its primary ratio depends on which of three rerun generations of four pairs with slow first epochs is kept (0.913 to 0.953), so we state no epoch advantage there.

End-to-end, the locality arm shows no detected advantage on any of the four reference graphs (pooled ratio 0.99). On IGB-hom-medium, it is slower in all 15 pairs, by 2.2 times overall and 1.9–2.6 times per cell ($p = 6 \times 10^{-5}$), because its ownership stage takes 54–61% of the run. At the measured epoch, saving the two arms break even only after about 120–200 epochs, against the 30 of the protocol. The warm-up therefore pays off in epoch time and gives no end-to-end gain on these graphs.

At 32 nodes, the locality epoch is shorter only on OGBN-Products and Reddit, in all five pairs each, a descriptive result with five pairs. The locality arm is slower end-to-end in 23 of 25 pairs, by up to 2.94 times on IGB-hom-medium, where its ownership stage is 60% of its end-to-end time against 49% on OGBN-Products. The advantage over DistDGL does not depend on the warm-up, since the modulo arm gives ratios of 0.25–0.44 at 32 nodes (Section 6.1).

No accuracy difference between the ownership maps is detected at 4–16 nodes, and every per-graph interval lies within $\pm 0.006$. At 32 nodes, the one unadjusted difference, $-0.0009$ on OGBN-Products, does not survive the correction over five graphs. Two further pair families, held at a fixed row count and at an equal ratio, completed at 16 workers, and their runs are in the repository. On OGBN-Products, the locality arm holds 21% fewer

owner rows per rank (one seed pair) at accuracy differences of 0.12 and 0.19 percentage points. On OGBN-Arxiv, the two counts differ by nine rows, so the saving appears where ownership coverage is partial. It is part of the controlled approximation of Section 3, so we do not count it as a gain. Section 7 lists the limits of this comparison.

### *6.7. Memory*

SoftPart needs less non-reclaimable host memory than DistDGL on four of the five graphs and more GPU memory on all but Reddit, so neither system holds a uniform memory advantage. Table 5 compares both systems like-for-like under the same job-scoped monitor (Section 5), per node over five seeds, and neither system has a uniform memory advantage.

**Table 5.** Per-node peak memory of both systems from the same job-scoped monitor (MiB; one worker per node; mean over five seeds of the maximum over nodes). Anonymous + shared is the peak of the job's anonymous plus shared memory, the host memory that the kernel cannot reclaim. PSS is the proportional set size of the job's processes, which also counts mapped file pages. The CUDA allocator peak is the device-side peak reported by the run. The measures are never summed. On IGB-hom-medium, the SoftPart runs requested twice the host memory of the runs of Table 3. DistDGL's partitioning job is outside the comparison (Section 6.7).

| | | **Anonymous + Shared** | | **Host PSS** | | **CUDA Allocator Peak** | |
|---|---|---|---|---|---|---|---|
| **Dataset** | **Nodes** | **SoftPart** | **DistDGL** | **SoftPart** | **DistDGL** | **SoftPart** | **DistDGL** |
| OGBN-Arxiv (bidir.) | 4 | 1344 | 2643 | 1677 | 2949 | 80 | 60 |
| | 8 | 1321 | 2673 | 1705 | 2980 | 62 | 54 |
| | 16 | 1310 | 2810 | 1688 | 3098 | 52 | 48 |
| Reddit | 4 | 2557 | 4780 | 3838 | 4854 | 405 | 778 |
| | 8 | 2376 | 4251 | 3620 | 4556 | 342 | 726 |
| | 16 | 2383 | 3913 | 3625 | 4226 | 311 | 715 |
| OGBN-Products | 4 | 2625 | 4256 | 4335 | 4483 | 377 | 283 |
| | 8 | 2573 | 3635 | 4272 | 3919 | 324 | 256 |
| | 16 | 2484 | 3390 | 4154 | 3714 | 274 | 214 |
| IGB-hom-small | 4 | 3103 | 4127 | 6964 | 4425 | 3069 | 681 |
| | 8 | 2463 | 3790 | 6499 | 4051 | 2158 | 564 |
| | 16 | 2212 | 3441 | 6484 | 3805 | 1359 | 463 |
| IGB-hom-medium | 4 | 26,425 | 22,733 | 54,987 | 22,812 | 25,884 | 739 |
| | 8 | 17,798 | 11,763 | 45,020 | 11,905 | 17,225 | 743 |
| | 16 | 13,076 | 7196 | 45,069 | 7048 | 12,155 | 654 |

**Non-reclaimable host memory.** SoftPart needs less of it than DistDGL in every run on four graphs, at 0.47–0.75 times DistDGL's. On IGB-hom-medium, it needs more in every run, from 1.16 times at four nodes to 1.82 at 16. Every one of these differences is significant after correction (Table 5).

**Host PSS.** SoftPart's PSS is lower than DistDGL's on OGBN-Arxiv, Reddit and OGBN-Products at four nodes and higher elsewhere, up to 6.4 times on IGB-hom-medium. SoftPart's exact evaluator projects the whole memory-mapped feature matrix on every rank, so on the IGB graphs its PSS contains nearly the whole feature file. These clean pages are reclaimable but are read again at every evaluation, so a smaller memory request costs time. Unlike DistDGL's runs, the SoftPart IGB-hom-medium runs of Table 5 requested twice the host memory per node of those of Table 3, so their host figures are not those of the reference sweep. At eight nodes, the doubled request shortens the end-to-end time by 8%.

**GPU memory.** SoftPart keeps its local feature store on the GPU and DistDGL keeps its features in host shared memory, so SoftPart's allocator peak is higher on every graph except Reddit (0.44–0.52 times DistDGL's). The gap is small on OGBN-Arxiv and OGBN-Products (1.1–1.3 times) and large on the IGB graphs, 2.9–4.5 times on IGB-hom-small and

19–35 times on IGB-hom-medium, up to 25 GiB per GPU. No worker copies the full feature matrix into private GPU memory.

These figures are one-second samples on shared nodes, where PSS divides file pages that a co-located job also maps, and they leave out DistDGL's partitioning, which runs in its own one-node job with a Slurm MaxRSS of 2–112 GiB across the graphs, while SoftPart's placement lies inside the measured run. Every SoftPart accuracy above comes from the evaluator of Section 4, which Section 6.8 checks against a single-machine reference.

### *6.8. Exactness of the Evaluator*

The implemented evaluator reproduces the analytic result to rounding level and changes no prediction once the matmul precision is pinned. Section 4.2 settles the analytic side, so we measured the numerical agreement of the implementation with a single-machine reference, the layer-wise inference path of DGL [9], at four ranks on four graphs. Two further checks compare four ranks against two on the same checkpoint and score models trained by DistDGL. In float32, the largest logit deviation is $1.22 \times 10^{-4}$ on SoftPart-trained mean models and $1.7 \times 10^{-4}$ on DistDGL-trained ones, and no prediction changes. The exception is the OGBN-Arxiv test scope, where the container's TF32 default changes five predictions and pinned float32 removes them. The sum aggregator deviates by up to 2.2 on its large logits and changes 25 predictions on OGBN-Arxiv at the default precision, none when pinned. The per-check records are in the repository.

The container's PyTorch build lets float32 products run on tensor cores, which round operands to 10 mantissa bits, and cuBLASLt chooses per shape. Against a float64 reconstruction, the OGBN-Arxiv test scope deviates by $7.95 \times 10^{-3}$ at the container default (`high`) and by $4.99 \times 10^{-6}$ at `highest` with no flip, so the deviation follows the matmul precision. A model of width $D = 256$, a case with $D > F$, shows the same pattern with two flips at `high` and none at `highest`. Two GPU evaluations of the same checkpoint are not bit-identical, but at pinned float32 their logits differ by at most $1.5 \times 10^{-5}$ with no changed prediction.

The 20 DistDGL-trained checkpoints (four graphs, five seeds, four-node runs of the second set) have no flipped test prediction at pinned float32 and deviate by about a float32 reconstruction's own deviation. Their evaluator accuracy equals the float64 accuracy and differs from DistDGL's own mean-of-partitions accuracy by at most $2.1 \times 10^{-5}$.

At pinned float32, the sum model has no flip on OGBN-Arxiv, Reddit or OGBN-Products. Its deviations from float64 reach 1.24 on its large logits and stay below those of a float32 reconstruction of the same model. The sum models reach 0.46, 0.86 and 0.67 against 0.64, 0.95 and 0.70 for the mean model, so the result shows exactness for a second aggregator and no competitive sum model. The gap is consistent with training on sums over at most the fan-out of neighbours and evaluating over all of them, which we did not test.

Every reported SoftPart accuracy was evaluated at `high`, and none of the checkpoints behind Table 3 were re-scored. Re-scoring at `highest` changes the test accuracy by $1.4 \times 10^{-4}$ for the OGBN-Arxiv sum model and by 0 for IGB-hom-small at four nodes, OGBN-Arxiv at $D = 256$ and the Reddit and OGBN-Products sum models. Training at `highest` changes the accuracy by at most $1.9 \times 10^{-3}$ per seed on three graphs at four nodes, within the seed-to-seed spread, and no interval excludes zero.

In 72 groups of separate jobs with the same graph, size, configuration and seed, the test accuracy differs by a median of $1.1 \times 10^{-4}$ and at most $4.2 \times 10^{-3}$, so same-seed training is not bit-reproducible. On OGBN-Arxiv, the largest ranges are of the size of the seed standard deviation. The accuracy differences of Section 6.3 thus rest on an evaluator whose own error stays at rounding level.

https://doi.org/10.3390/a19100900

Taken together, the placement pays for itself on every graph at four to 16 nodes, the warm-up shortens the training step on four graphs, SoftPart needs less non-reclaimable host memory than DistDGL on four of the five graphs, and the evaluator behind every number holds to rounding level. What grows with the graph is the placement share of the run, and Section 7 states this together with the other limits.

## 7. Limitations

The measured model is a two-layer mean-aggregating GraphSAGE, with the sum aggregator as a second case. The evaluator identity does not hold for max or attention aggregation (Section 4.2), and deeper models would need a layer-wise evaluator and a generalised sampler (Section 8). The cache stays static after the warm-up, and the measurements cover graphs of 0.17 to 10 M vertices on 4–32 nodes with one GPU each. The hidden width is 16 in all sweeps, which keeps the gradient exchange small. Widths of 64 and 256, run on OGBN-Arxiv at eight nodes only, reach 0.671 and 0.687 against 0.630 at width 16, so on that graph the reference width costs accuracy. DistDGL was not run above width 16, so the accuracy comparison holds at width 16 only.

**Protocol.** DistDGL's primary end-to-end time starts from a pre-built METIS partition, so our primary ratios are upper bounds. Counting the partitioning takes the twelve ratios to 0.21–0.35 (Section 6.1). DistDGL ran in one fixed configuration on the `dgl.distributed` path without GraphBolt, and its six further configurations cover two graphs at eight nodes (Section 6.5).

**PyG baseline.** A second baseline on the maintained PyG path (cuGraph-PyG with WholeGraph feature storage) was measured under the same configuration on the five graphs at four, eight and 16 nodes, five seeds per cell. SoftPart's Slurm end-to-end time is 0.10 to 0.35 of PyG's and DistDGL's 0.40 to 0.95 of it, and PyG is the slowest of the three systems in every cell. PyG's test accuracy is within 0.0025 of SoftPart's on Reddit, OGBN-Products, IGB-hom-small and IGB-hom-medium and 0.005 to 0.016 below it on OGBN-Arxiv, where DistDGL shows a similar gap. The PyG measurements need further investigation. We could not measure $P^3$ [4] because its authors released no implementation and the two third-party reimplementations target a single multi-GPU host [61] and a superseded DGL interface [62], so Section 2 places it by its published numbers.

**Verification.** The agreement with a single-machine reference is measured at four ranks on four graphs for the mean aggregator and on three graphs for the sum aggregator (Section 6.8). Re-scoring at full float32 covers only the checkpoints named there.

**Flagged and outlier runs.** Two of the sixty DistDGL runs are flagged as accuracy-anomalous, one on IGB-hom-small and one on OGBN-Arxiv (bidir.), both at four nodes. A flag requires an `srun` warning in the run's standard error and a final test accuracy more than five standard deviations from the mean of the other four runs. We keep every run and mark those cells ‡ in Table 3. One DistDGL run was replaced after a concurrent job on a shared node overwrote its DGL shared-memory tensors [63], because its logged class count and split sizes no longer matched the dataset. The replacement used the same settings and partition seed.

**Placement coverage and the controlled approximation.** Warm-up stops at a coverage target defined on the training seeds only, so vertices outside the seeds that no warm-up batch touched stay unowned. They enter training only through a cache, when the shortfall score of a rank's own rows ranks them in, and otherwise are never sampled (Section 6.5).

**Design partly isolated.** The paired control holds the budget at $f = 0.25$ (Section 6.6), and the budget is varied only in a separate strict-policy sweep at 16 nodes. Budgets between 10% of the owned rows and the published policy were not measured.

**No distributed owner directory.** The claim lists and a graph-sized owner map are replicated on every rank, and at a fixed number of checkpoints the all-gather traffic grows as $W^2$ times the per-rank claim volume. At 10 M vertices, the owner-map build is 78% of the ownership stage, which is 54–61% of the run. A distributed owner directory is not implemented, so this design limit is met first as graphs or world sizes grow.

**Deficit at the largest sizes.** SoftPart is less accurate after correction on Reddit at eight and 16 nodes and on IGB-hom-small and IGB-hom-medium at 16 nodes, by at most 0.0027. At 32 nodes, the Reddit deficit is 0.0042 and the 0.006 band does not hold there and is left open on OGBN-Arxiv, IGB-hom-small is lower by 0.0024 and non-inferior, and, on OGBN-Products, the locality ownership's advantage over DistDGL falls short of significance (ratio 0.91). The restricted local view is a candidate source that we have not isolated (Section 6.4).

**Measured range.** The largest graph has 10 M vertices and the largest world size is 32 nodes with one GPU each. Graphs of order $10^8$ vertices exceed the available hardware, since the baseline's partitioning job alone needs 112 GiB at 10 M vertices. With one GPU per node, the partition allows at most 58 workers, so 64 workers were not run. Larger scales rest on the closed-form extrapolation of Section 6.2, and the conclusions hold for the measured range.

Section 8 turns two of these limits, the replicated owner map and the two-layer depth, into future work.

## 8. Conclusions and Future Work

In this paper, we built and measured SoftPart GNN, a distributed, sampled GraphSAGE trainer that runs without an offline partitioner. A short workload-driven warm-up, a deterministic soft ownership and a budgeted foreign-vertex cache give every rank a local view, so training needs no per-minibatch remote access and its only per-step communication is gradient synchronisation. Model quality was measured with an evaluator that does not depend on the training layout and is exact in exact arithmetic. DistDGL's models are scored by DGL's own full-neighbour inference, and the two agree with a single-machine reference to the rounding level where we verified them.

At four to 16 nodes, SoftPart finishes in 0.22–0.62 of DistDGL's time on five graphs, 0.21–0.52 with the partitioning counted, and every SoftPart run is faster than every DistDGL run of its cell. On four graphs at 32 nodes, the ratio is 0.40–0.91, and on OGBN-Products, the difference is not significant. SoftPart is also faster in every run against six tuned DistDGL configurations on two graphs at eight nodes and against the PyG baseline on five graphs at four to 16 nodes. The primary ratio excludes the baseline's partitioning, so its advantage is consistent with the absence of per-minibatch remote access, does not depend on the warm-up and widens once the partitioning is counted. At four to 16 nodes, accuracy stays within 0.006 of DistDGL's on four graphs, with OGBN-Arxiv left open. It is lower after correction in six cells, four at eight or 16 nodes by at most 0.0027 and two at 32 nodes. At 32 nodes, Reddit falls outside the 0.006 band and OGBN-Arxiv is again left open (Section 6).

A paired control that replaces the warm-up with $o(v) = v \bmod W$ and holds the cache policy and budget fixed (Section 6.6) detects no accuracy difference. Without the warm-up, the training epoch at four to 16 nodes is longer on four of the five graphs, by 11 to 52 percent as per-graph geometric means, with no detected difference on OGBN-Arxiv. The end-to-end time is still not shorter with the warm-up on any graph, because placement has to be paid for first. Placement takes a fifth to over a half of the run on the smaller graphs and more than half at ten million vertices (Section 6.2). Memory is not a uniform advantage. SoftPart needs less non-reclaimable host memory than the baseline on four graphs and

more on the largest, and more GPU memory except on Reddit, because its feature store is GPU-resident.

Our plans for future work follow from two limits that the measurements made explicit. Ownership is the placement component whose share of the run grows with the world size, and the owner-map build is its largest term on the larger graphs. The first direction is therefore a distributed owner directory, which would replace the all-gather claim exchange and the per-rank graph-sized owner map, and which we discuss and did not measure. Distributed graph stores and key–value services already play that role in partitioned trainers [1–3]. The second concerns depth, because the evaluator and the native sampler are two-layer (Section 4). A $k$-layer exact evaluator following the layer-wise inference pattern [9] would lift the evaluator's two-layer limit, and the sampler and its block interface would have to be generalised with it.

The measurements answer the question of the title for two-layer GraphSAGE on graphs of up to ten million vertices. A soft placement derived from the workload at runtime replaces offline partitioning, trains in a fraction of DistDGL's time and reaches its accuracy within 0.006 on four of the five graphs, and the evaluator behind these numbers is exact to rounding level. Every step of that comparison is measured, and the records are released with the code.

**Supplementary Materials:** The following supporting information can be downloaded at https://www.mdpi.com/article/10.3390/a19100900/s1, Table S1: the 49 GNN training systems of Section 2.3 with their layout labels and evidence locations; Table S2: Cache budget at 16 nodes; Table S3: Fan-out sensitivity of both systems at 8 nodes; Table S4: DistDGL with tuned system settings at 8 nodes.

**Author Contributions:** Conceptualisation, A.F.; methodology, A.F. and M.N.; software, M.N.; validation, M.N. and A.F.; formal analysis, A.F.; investigation, M.N. and A.F.; resources, A.F.; data curation, M.N. and A.F.; writing—original draft preparation, A.F.; writing—review and editing, A.F., M.N. and G.K.; visualisation, A.F.; supervision, G.K.; project administration, A.F.; funding acquisition, A.F. and G.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the Hungarian AI Factory Antenna (HunAIFA) project under the EuroHPC Joint Undertaking's AI Factory Antennas programme (grant agreement No 101263244).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The five graphs used in this study are publicly available, namely OGBN-Arxiv and OGBN-Products from the Open Graph Benchmark [54], Reddit as released with GraphSAGE [32] and IGB-hom-small and IGB-hom-medium from the Illinois Graph Benchmark [55]. The per-run and per-cell records of both sweeps behind Tables 3 and 5 and Figures 2–4, the DistDGL reference runs, the paired control runs behind Table 4, the PyG baseline runs of Section 7, the exactness-verification runs of Section 6.8, the analysis and figure scripts, the container definition of Section 5.1 and the SoftPart GNN source code are available in the GitHub repository at https://github.com/fsattila/SoftPart-GNN (accessed on 15 October 2026; release v1.1.0, with a reproduction guide) under the Apache-2.0 licence.

**Acknowledgments:** The development work was supported by the HUN-REN Cloud (https://science-cloud.hu/, accessed on 15 October 2026) within the project "Distributed deep learning in parallel and distributed environment". We acknowledge the Digital Government Development and Project Management Ltd. for awarding us access to the Komondor HPC facility based in Hungary. The authors thankfully acknowledge the continuous support of the Doctoral School of Applied Informatics and Applied Mathematics at Obuda University. During the preparation of this manuscript, the authors used Claude (Anthropic; Claude Opus 5 and 5.5, via Claude Science and Claude Code) for literature screening and relevance labelling, for drafting and revising text passages, for language editing, and for scripting the analysis and figure generation, and OpenAI Codex (OpenAI; models

gpt-5.5 and gpt-5.6-sol) for implementation support, experiment orchestration and result verification. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| GNN | Graph neural network |
| DDP | Distributed data-parallel |
| E2E | End-to-end |
| CSR | Compressed sparse row |
| DGL | Deep Graph Library |
| RSS | Resident set size |
| CPU | Central processing unit |
| GPU | Graphics processing unit |
| TF32 | TensorFloat-32 |

## Appendix A. Claim Resolution and Cache Selection

**Algorithm A1** claim resolution, RESOLVE

**Input:** own claim set $C_w$, seed $s$; executed identically on every rank
**Hashes:** $\textsc{Hash}(v, s)$ is the mixing function of Section 3.6 with $s_r = 0$; HASHSAMPLE is a separate deterministic hash, both fixed by the released implementation.

1: $(C_0, \ldots, C_{W-1}) \leftarrow \textsc{AllGatherObject}(\text{sorted } C_w)$
2: **for** each $v$ with claimant list $L_v \neq \emptyset$, ordered by rank **do**
3: $\quad o(v) \leftarrow L_v[\,\textsc{Hash}(v, s) \bmod |L_v|\,]$
4: **end for**
5: **for** each $v$ with $L_v = \emptyset$ **do**
6: $\quad o(v) \leftarrow \perp$
7: **end for**
8: **return** $o$

For a row with $d \leq k$, the scoring pass is provably a no-op, since, there, $t = d$ and $\min(|\mathcal{N}_{\text{loc}}|, k) = |\mathcal{N}_{\text{loc}}|$, so `shortfall` $= |\mathcal{N}_{\text{for}}|$ and `cand` $= \mathcal{N}_{\text{for}}$, so every scored candidate of a low-degree row is already in $M_w$ and is removed from `ordered` again. The mandatory base and the score therefore do not double-count.

**Algorithm A2** degree/shortfall cache selection

**Input:** $G$, owned seeds $O_w \cap S$, ownership $o$, rank $w$ (policy `degree_shortfall_random_fast`)
last fan-out $k = k_2$, row budget $b$, seed $s$

1: $M_w \leftarrow \emptyset$ ▷ mandatory base
2: `score`$[u] \leftarrow 0$ for every $u$

**Algorithm A2** Cont.

**for** $v \in O_w \cap S$ **do**
  $\mathcal{N}_{\text{loc}} \leftarrow \{u \in \mathcal{N}(v) : o(u) = w\}$; $\mathcal{N}_{\text{for}} \leftarrow$ the adjacency entries of $v$ not owned by $w$, as stored (a multiset); $d \leftarrow |\mathcal{N}(v)|$
  **if** $d \le k$ **then**
    $M_w \leftarrow M_w \cup \mathcal{N}_{\text{for}}$ ▷ low-degree row: every neighbour is used
  **end if**
  $t \leftarrow \min(d, k)$; $\texttt{shortfall} \leftarrow \max\left(0,\ t - \min(|\mathcal{N}_{\text{loc}}|, k)\right)$
  **if** $\texttt{shortfall} = 0$ **then**
    **continue**
  **end if**
  $\texttt{cand} \leftarrow \textsc{HashSample}(\mathcal{N}_{\text{for}},\ \min(\texttt{shortfall}, |\mathcal{N}_{\text{for}}|),\ s, w, v)$
  **for** $u \in \texttt{cand}$ **do**
    $\texttt{score}[u] \leftarrow \texttt{score}[u] + 1$
  **end for**
**end for**
$\texttt{ordered} \leftarrow$ the scored vertices outside $M_w$, sorted by descending score, then by ascending vertex id
$r \leftarrow \max(0,\ b - |M_w|)$
$K_w \leftarrow M_w \cup \{$the first $r$ elements of $\texttt{ordered}\}$
**return** $K_w$

## References


1. Zheng, D.; Ma, C.; Wang, M.; Zhou, J.; Su, Q.; Song, X.; Gan, Q.; Zhang, Z.; Karypis, G. DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs. In *Proceedings of the 2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*; IEEE: New York, NY, USA, 2020; pp. 36–44. [CrossRef]
2. Zhu, R.; Zhao, K.; Yang, H.; Lin, W.; Zhou, C.; Ai, B.; Li, Y.; Zhou, J. AliGraph: A Comprehensive Graph Neural Network Platform. *Proc. VLDB Endow.* **2019**, *12*, 2094–2105. [CrossRef]
3. Zheng, C.; Chen, H.; Cheng, Y.; Song, Z.; Wu, Y.; Li, C.; Cheng, J.; Yang, H.; Zhang, S. ByteGNN: Efficient Graph Neural Network Training at Large Scale. *Proc. VLDB Endow.* **2022**, *15*, 1228–1242. [CrossRef]
4. Gandhi, S.; Iyer, A.P. P3: Distributed Deep Graph Learning at Scale. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*; USENIX Association: Berkeley, CA, USA, 2021; pp. 551–568.
5. Mu, Z.; Tang, S.; Zhuang, Y.; Yu, D. Attribute-Driven Streaming Edge Partitioning with Reconciliations for Distributed Graph Neural Network Training. *Neural Netw.* **2023**, *165*, 987–998. [CrossRef] [PubMed]
6. Polisetty, S.; Liu, J.; Fung, Y.; Lim, S.H.; Guan, H.; Serafini, M. Spa: Scaling Graph Neural Network Training on Large Graphs via Probabilistic Splitting. In Proceedings of the Eighth Conference on Machine Learning and Systems (MLSys 2025), Santa Clara, CA, USA, 12–15 May 2025.
7. Waleffe, R.; Sarda, D.; Mohoney, J.; Vlatakis-Gkaragkounis, E.V.; Rekatsinas, T.; Venkataraman, S. Armada: Memory-Efficient Distributed Training of Large-Scale Graph Neural Networks. *arXiv* **2025**, arXiv:2502.17846.
8. Merkel, N.; Stoll, D.; Mayer, R.; Jacobsen, H.A. An Experimental Comparison of Partitioning Strategies for Distributed Graph Neural Network Training. In Proceedings of the 28th International Conference on Extending Database Technology (EDBT 2025), Barcelona, Spain, 25–28 March 2025. [CrossRef]
9. Wang, M.; Zheng, D.; Ye, Z.; Gan, Q.; Li, M.; Song, X.; Zhou, J.; Ma, C.; Yu, L.; Gai, Y.; et al. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv* **2019**, arXiv:1909.01315.
10. Ma, L.; Yang, Z.; Miao, Y.; Xue, J.; Wu, M.; Zhou, L.; Dai, Y. NeuGraph: Parallel Deep Neural Network Computation on Large Graphs. In Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC 19), Renton, WA, USA, 10–12 July 2019; pp. 443–458.
11. Peng, J.; Chen, Z.; Shao, Y.; Shen, Y.; Chen, L.; Cao, J. SANCUS: Staleness-Aware Communication-Avoiding Full-Graph Decentralized Training in Large-Scale Graph Neural Networks. *Proc. VLDB Endow.* **2022**, *15*, 1937–1950. [CrossRef]
12. Wang, Q.; Zhang, Y.; Wang, H.; Chen, C.; Zhang, X.; Yu, G. NeutronStar: Distributed GNN Training with Hybrid Dependency Management. In *Proceedings of the 2022 International Conference on Management of Data*; ACM: New York, NY, USA, 2022; pp. 1301–1315. [CrossRef]

13. Yang, S.; Zhang, M.; Dong, W.; Li, D. Betty: Enabling Large-Scale GNN Training with Batch-Level Graph Partitioning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*; ACM: New York, NY, USA, 2023; Volume 2, pp. 103–117. [CrossRef]
14. Wang, Q.; Chen, Y.; Wong, W.F.; He, B. HongTu: Scalable Full-Graph GNN Training on Multiple GPUs. *Proc. ACM Manag. Data* **2023**, *1*, 1–27. [CrossRef]
15. Cai, Z.; Zhou, Q.; Yan, X.; Zheng, D.; Song, X.; Zheng, C.; Cheng, J.; Karypis, G. DSP: Efficient GNN Training with Multiple GPUs. In Proceedings of the ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, Montreal, QC, Canada, 25 February–1 March 2023; pp. 392–404. [CrossRef]
16. Chen, W.; He, S.; Qu, H.; Zhang, X. LeapGNN: Accelerating Distributed GNN Training Leveraging Feature-Centric Model Migration. In Proceedings of the 23rd USENIX Conference on File and Storage Technologies (FAST 25), Santa Clara, CA, USA, 25–27 February 2025; pp. 255–270.
17. Liu, R.; Wang, Y.; Yan, X.; Jiang, H.; Cai, Z.; Wang, M.; Tang, B.; Li, J. DiskGNN: Bridging I/O Efficiency and Model Accuracy for Out-of-Core GNN Training. *Proc. ACM Manag. Data* **2025**, *3*, 1–27. [CrossRef]
18. Zhang, Z.; Wu, C. Online Graph Partition for Distributed Dynamic GNN Training. *ACM Trans. Model. Perform. Eval. Comput. Syst.* **2025**, *10*, 1–23. [CrossRef]
19. Lin, Z.; Li, C.; Miao, Y.; Liu, Y.; Xu, Y. PaGraph: Scaling GNN Training on Large Graphs via Computation-Aware Caching. In *Proceedings of the 11th ACM Symposium on Cloud Computing*; ACM: New York, NY, USA, 2020; pp. 401–415. [CrossRef]
20. Park, Y.; Min, S.; Lee, J.W. Ginex: SSD-enabled Billion-scale Graph Neural Network Training on a Single Machine via Provably Optimal In-memory Caching. *Proc. VLDB Endow.* **2022**, *15*, 2626–2639. [CrossRef]
21. Sheng, Z.; Zhang, W.; Tao, Y.; Cui, B. OUTRE: An OUT-of-Core De-REdundancy GNN Training Framework for Massive Graphs within A Single Machine. *Proc. VLDB Endow.* **2024**, *17*, 2960–2973. [CrossRef]
22. Sarkar, A.; Ghosh, S.; Tallent, N.R.; Jannesari, A. MassiveGNN: Efficient Training via Prefetching for Massively Connected Distributed Graphs. In *Proceedings of the 2024 IEEE International Conference on Cluster Computing (CLUSTER)*; IEEE: New York, NY, USA, 2024; pp. 62–73. [CrossRef]
23. Liu, T.; Chen, Y.; Li, D.; Wu, C.; Zhu, Y.; He, J.; Peng, Y.; Chen, H.; Chen, H.; Guo, C. BGL: GPU-Efficient GNN Training by Optimizing Graph Data I/O and Preprocessing. In Proceedings of the Symposium on Networked Systems Design and Implementation, Boston, MA, USA, 17–19 April 2023; pp. 103–118.
24. Sun, J.; Su, L.; Shi, Z.; Shen, W.; Wang, Z.; Wang, L.; Zhang, J.; Li, Y.; Yu, W.; Zhou, J.; et al. Legion: Automatically Pushing the Envelope of Multi-GPU System for Billion-Scale GNN Training. In Proceedings of the USENIX Annual Technical Conference, Boston, MA, USA, 10–12 July 2023; pp. 165–179.
25. Yang, D.; Liu, J.; Qi, J.; Lai, J. WholeGraph: A Fast Graph Neural Network Training Framework with Multi-GPU Distributed Shared Memory Architecture. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, 13–18 November 2022; pp. 1–14. [CrossRef]
26. Yang, J.; Tang, D.; Song, X.; Wang, L.; Yin, Q.; Chen, R.; Yu, W.; Zhou, J. GNNLab: A Factored System for Sample-Based GNN Training over GPUs. In Proceedings of the European Conference on Computer Systems, Rennes, France, 5–8 April 2022; pp. 417–434. [CrossRef]
27. Kaler, T.; Iliopoulos, A.S.; Murzynowski, P.; Schardl, T.B.; Leiserson, C.E.; Chen, J. Communication-Efficient Graph Neural Networks with Probabilistic Neighborhood Expansion Analysis and Caching. In Proceedings of the Conference on Machine Learning and Systems, Miami, FL, USA, 4–8 June 2023.
28. Huang, X.; Lee, C.H. Demystifying Distributed Training of Graph Neural Networks for Link Prediction. In *Proceedings of the 2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS)*; IEEE: New York, NY, USA, 2025; pp. 199–209. [CrossRef]
29. Fey, M.; Lenssen, J.E.; Weichert, F.; Leskovec, J. GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings. In Proceedings of the 38th International Conference on Machine Learning (ICML 2021), Online, 18–24 July 2021.
30. Wan, C.; Li, Y.; Li, A.; Kim, N.S.; Lin, Y. BNS-GCN: Efficient Full-Graph Training of Graph Convolutional Networks with Partition-Parallelism and Random Boundary Node Sampling. In Proceedings of the Conference on Machine Learning and Systems, Santa Clara Convention Center, Santa Clara, CA, USA, 29 August–1 September 2022.
31. Jia, Z.; Lin, S.; Gao, M.; Zaharia, M.; Aiken, A. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with ROC. In Proceedings of the Machine Learning and Systems 2 (MLSys 2020), Austin, TX, USA, 2–4 March 2020; pp. 187–198.
32. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive Representation Learning on Large Graphs. In Proceedings of the Advances in Neural Information Processing Systems 30 (NeurIPS 2017), Long Beach, CA, USA, 4–9 December 2017.
33. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Proceedings of the Advances in Neural Information Processing Systems 32, Vancouver, BC, Canada, 8–14 December 2019.

34. Li, S.; Zhao, Y.; Varma, R.; Salpekar, O.; Noordhuis, P.; Li, T.; Paszke, A.; Smith, J.; Vaughan, B.; Damania, P.; et al. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proc. VLDB Endow.* **2020**, *13*, 3005–3018. [CrossRef]
35. Gonzalez, J.E.; Low, Y.; Gu, H.; Bickson, D.; Guestrin, C. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*; USENIX Association: Berkeley, CA, USA, 2012; pp. 17–30.
36. Karypis, G.; Kumar, V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* **1998**, *20*, 359–392. [CrossRef]
37. Chen, J.; Ma, T.; Xiao, C. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
38. 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 Proceedings of the Advances in Neural Information Processing Systems 32, Vancouver, BC, Canada, 8–14 December 2019.
39. Karger, D.; Lehman, E.; Leighton, T.; Panigrahy, R.; Levine, M.; Lewin, D. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC '97)*; ACM: New York, NY, USA, 1997; pp. 654–663. [CrossRef]
40. Malewicz, G.; Austern, M.H.; Bik, A.J.; Dehnert, J.C.; Horn, I.; Leiser, N.; Czajkowski, G. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 2010; pp. 135–146. [CrossRef]
41. Vasimuddin; Misra, S.; Ma, G.; Mohanty, R.; Georganas, E.; Heinecke, A.; Kalamkar, D.D.; Ahmed, N.; Avancha, S. DistGNN: Scalable Distributed Training for Large-Scale Graph Neural Networks. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, St. Louis, MO, USA, 14–19 November 2021; pp. 1–14. [CrossRef]
42. Wan, C.; Li, Y.; Wolfe, C.R.; Kyrillidis, A.; Kim, N.S.; Lin, Y. PipeGCN: Efficient Full-Graph Training of Graph Convolutional Networks with Pipelined Feature Communication. In Proceedings of the International Conference on Learning Representations, Online, 25–29 April 2022.
43. 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 *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*; ACM: New York, NY, USA, 2019; pp. 257–266. [CrossRef]
44. Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; Prasanna, V. GraphSAINT: Graph Sampling Based Inductive Learning Method. In Proceedings of the International Conference on Learning Representations, Online, 26 April–1 May 2020.
45. Hockney, R.W. The Communication Challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Comput.* **1994**, *20*, 389–398. [CrossRef]
46. Thakur, R.; Rabenseifner, R.; Gropp, W. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* **2005**, *19*, 49–66. [CrossRef]
47. Goldberg, D. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Comput. Surv.* **1991**, *23*, 5–48. [CrossRef]
48. Thorpe, J.; Qiao, Y.; Eyolfson, J.; Teng, S.; Hu, G.; Jia, Z.; Wei, J.; Vora, K.; Netravali, R.; Kim, M.; et al. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, Online, 14–16 July 2021.
49. Chen, S.; Song, X.; Vasiloudis, T.; Liu, H. Deal: Distributed End-to-End GNN Inference for All Nodes. *arXiv* **2025**, arXiv:2503.02960.
50. Digital Government Development and Project Management Ltd. Komondor HPC Documentation: Hardware Architecture and GPU Partition. 2025. Available online: https://docs.hpc.dkf.hu/hardware/hardware.html (accessed on 10 September 2026).
51. Yoo, A.B.; Jette, M.A.; Grondona, M. SLURM: Simple Linux Utility for Resource Management. In *Proceedings of the Job Scheduling Strategies for Parallel Processing (JSSPP 2003)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2862, pp. 44–60. [CrossRef]
52. PyTorch Contributors. Gloo: A Collective Communications Library. 2026. Available online: https://github.com/pytorch/gloo (accessed on 6 September 2026).
53. Kurtzer, G.M.; Sochat, V.; Bauer, M.W. Singularity: Scientific Containers for Mobility of Compute. *PLoS ONE* **2017**, *12*, e0177459. [CrossRef] [PubMed]
54. 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 Proceedings of the Advances in Neural Information Processing Systems 33, Online, 6–12 December 2020.
55. Khatua, A.; Mailthody, V.S.; Taleka, B.; Ma, T.; Song, X.; Hwu, W. IGB: Addressing the Gaps in Labeling, Features, Heterogeneity, and Size of Public Graph Datasets for Deep Learning Research. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 2023; pp. 4284–4295. [CrossRef]
56. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.

57. DGL Team. Chapter 7: Distributed Training — DGL Documentation. 2026. Available online: https://www.dgl.ai/dgl_docs/guide/distributed.html (accessed on 6 September 2026).
58. Nagy, M.; Farkas, A.; Kertész, G. Distributed Deep Graph Learning with DistDGL: A Case Study in a Cloud Environment. In Proceedings of the 2025 IEEE 25th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 18–20 November 2025; pp. 241–246. [CrossRef]
59. Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv* **2017**, arXiv:1706.02677.
60. Keskar, N.S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; Tang, P.T.P. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
61. Liu, J. P3-GNN: Implementation for "P3: Distributed Deep Graph Learning at Scale". 2023. GitHub Repository, MIT Licence. Available online: https://github.com/juelinl/P3-GNN (accessed on 8 September 2026).
62. ISCS-ZJU. LeapGNN-AE: Artifact of LeapGNN, Branch `distributed_version`, 2025. GitHub Repository; Contains `dgl_p3.py` and `graphsage_p3.py`. Available online: https://github.com/ISCS-ZJU/LeapGNN-AE (accessed on 8 September 2026).
63. Emődi, M.B.; Kovács, J.; Lovas, R. Mesh-aware debugging: Identifying resource allocation issues in distributed microservices. *Acta Polytech. Hung.* **2025**, *22*, 9–25. [CrossRef]


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.