NeutronOrch: **Rethinking** Sample-based **GNN Training** under **CPU-GPU** Heterogeneous Environments

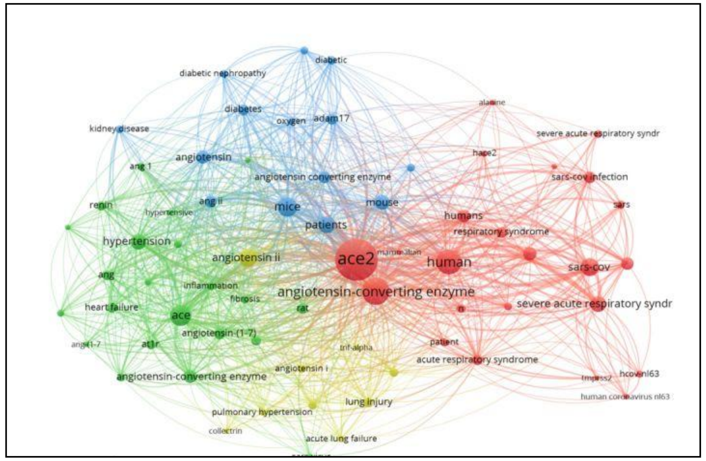
Xin Ai, Qiange Wang, Chunyu Cao, Yanfeng Zhang, Chaoyi Chen, Hao Yuan, Yu Gu, Ge Yu

School of Computer Science and Engineering

Northeastern University, Shenyang, China

Graph Neural Network

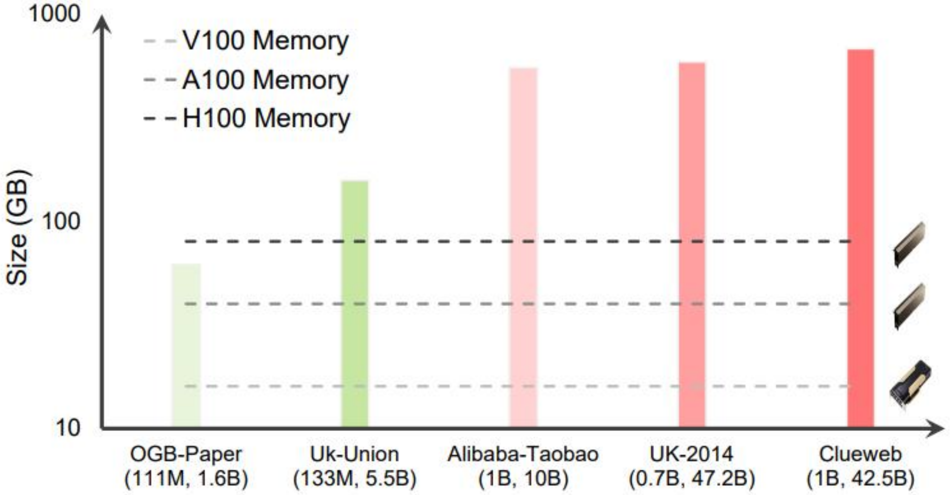
(a) Social Networks



(b) Knowledge Graph

(c) Biological networks

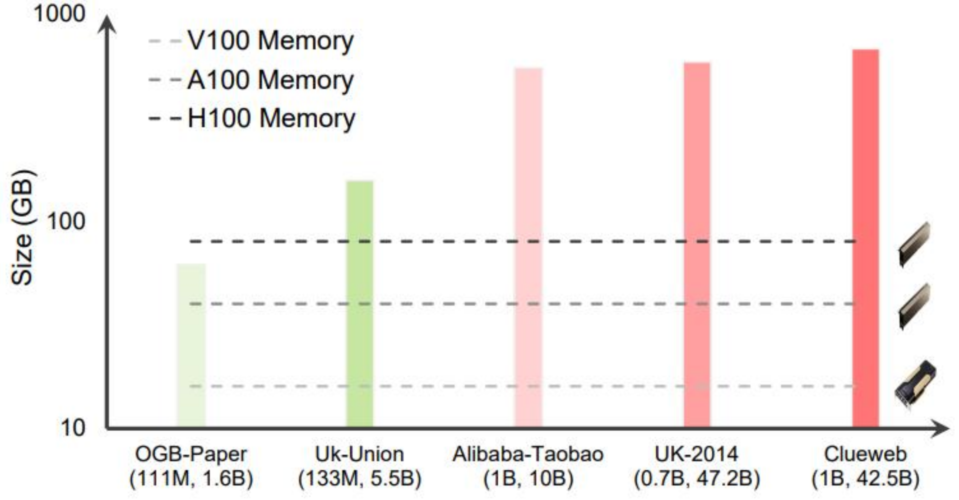
Challeng from Industry



GNN dataset size and current GPU capacities [Legion:ATC'23]

☹ hard to scale to large graphs

Challeng from Industry



GNN dataset size and current GPU capacities [Legion:ATC’23]

☹ hard to scale to large graphs

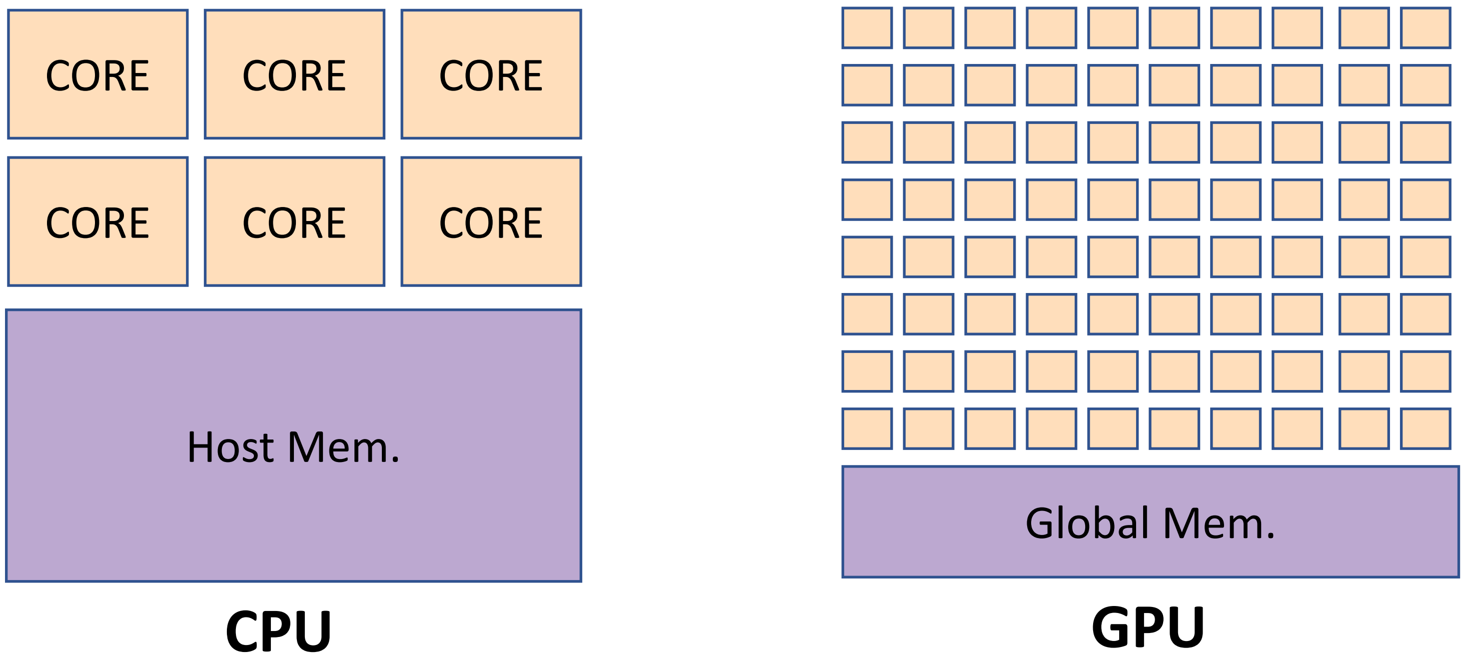
⇩

CPU-GPU Heterogeneous Platforms

+

Sampling-based GNN Training

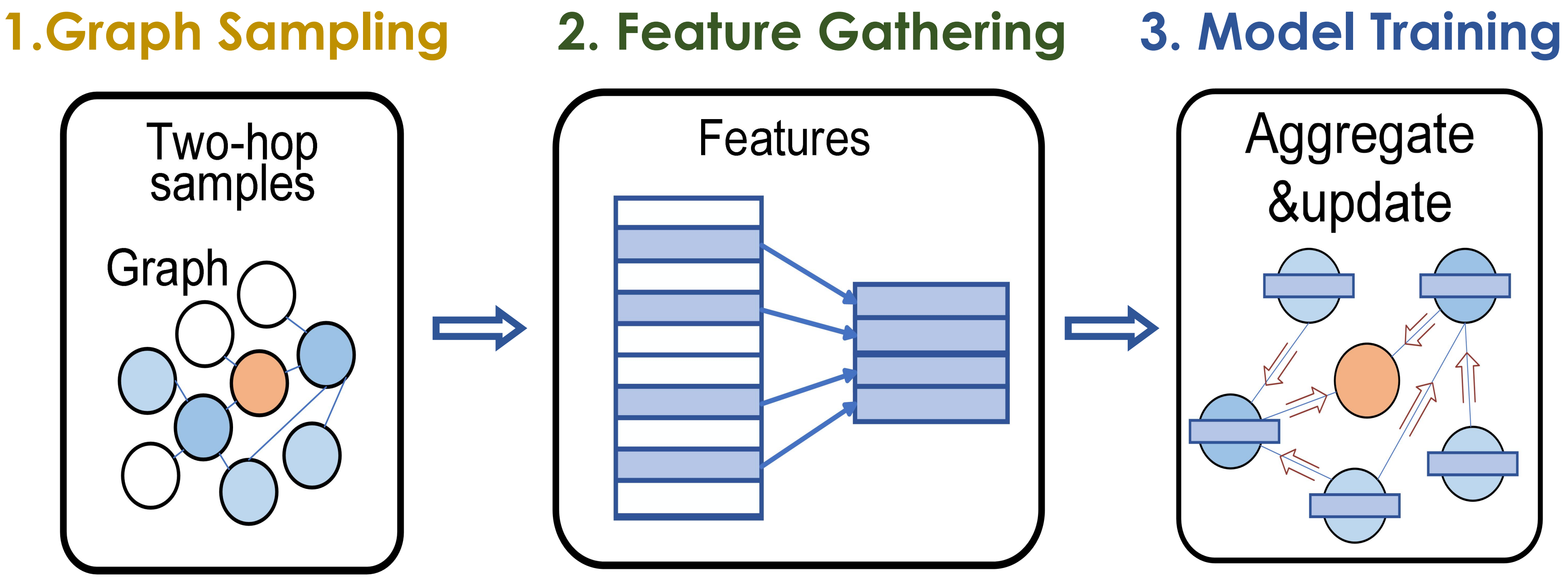
CPU and GPU



- CPU： Large Memory Capacity(main memory)； Low Parallelism
- GPU： Limited Memeory Capacity； High Parallelism

Sampling-based GNN

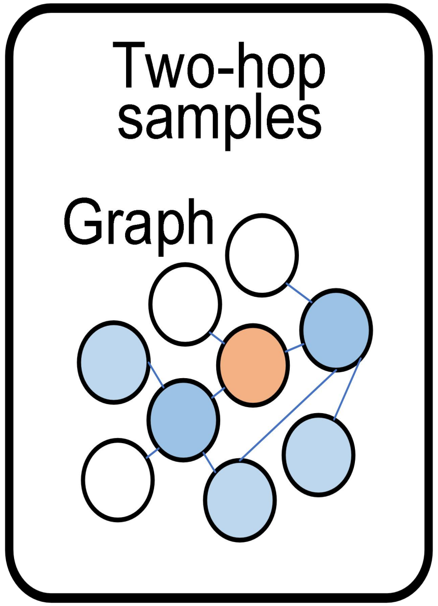
- **Three Key Steps:**



Sampling-based GNN

- **Three Key Steps:**

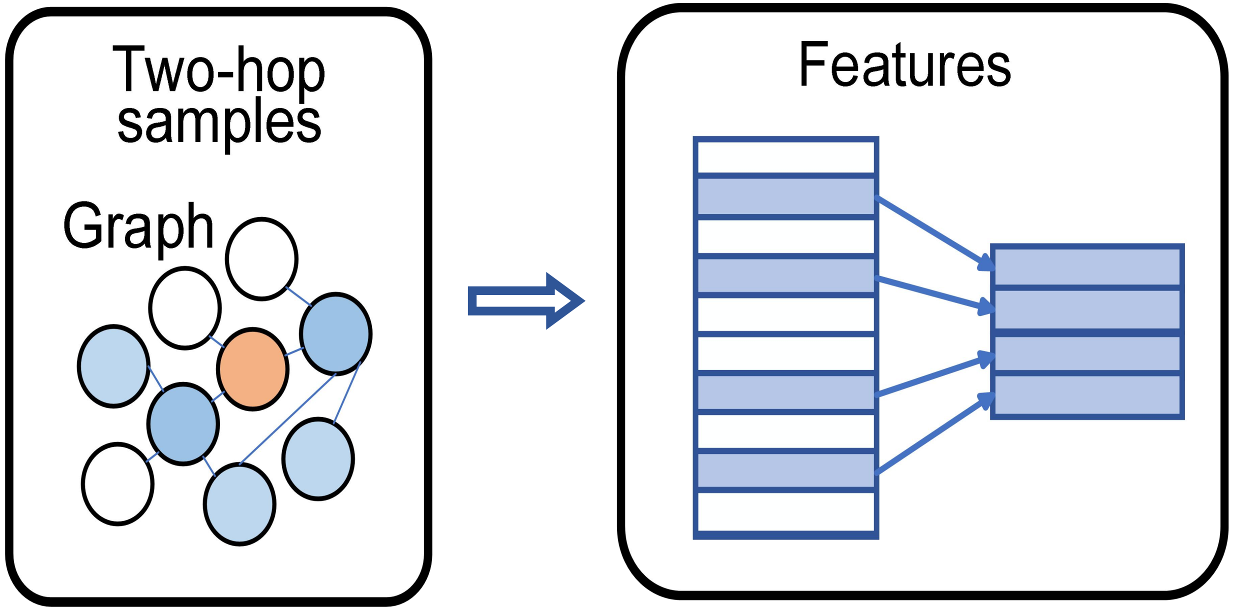
1.Graph Sampling　　2. Feature Gathering　　3. Model Training



Sampling-based GNN

- **Three Key Steps:**

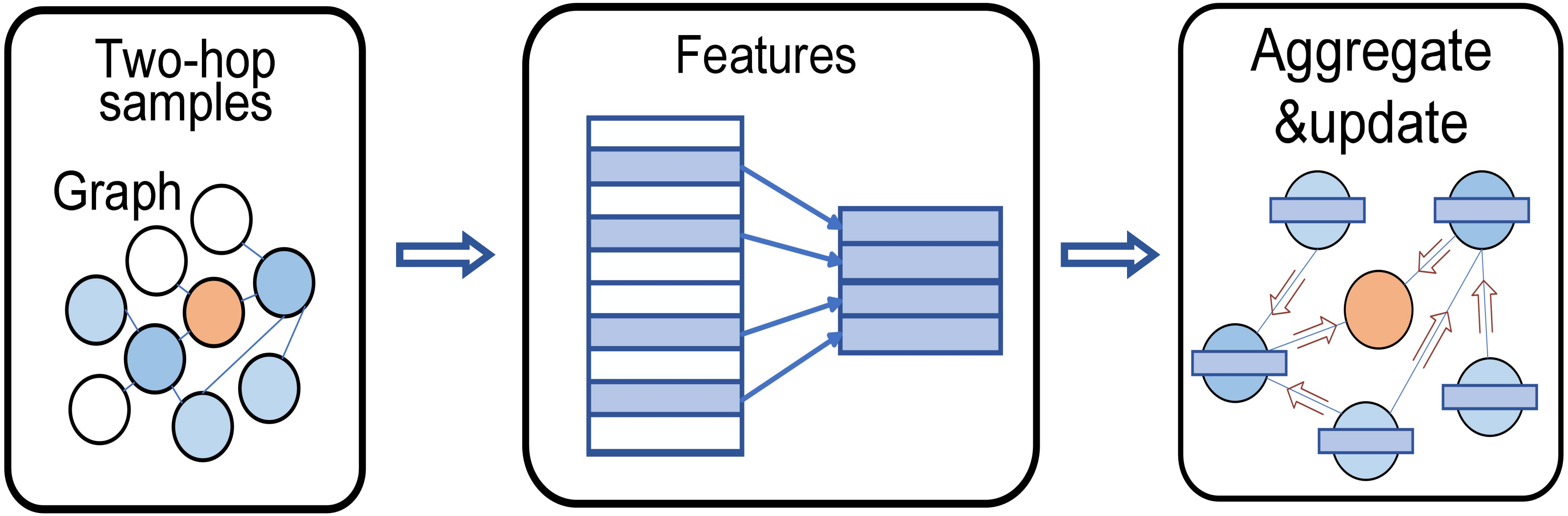
1.Graph Sampling **2. Feature Gathering** **3. Model Training**



Sampling-based GNN

- **Three Key Steps:**

1.Graph Sampling **2. Feature Gathering** **3. Model Training**



Existing GNN Systems

Step-based task orchestrating methods

1.Graph Sampling

2. Feature Gathering

3. Model Training

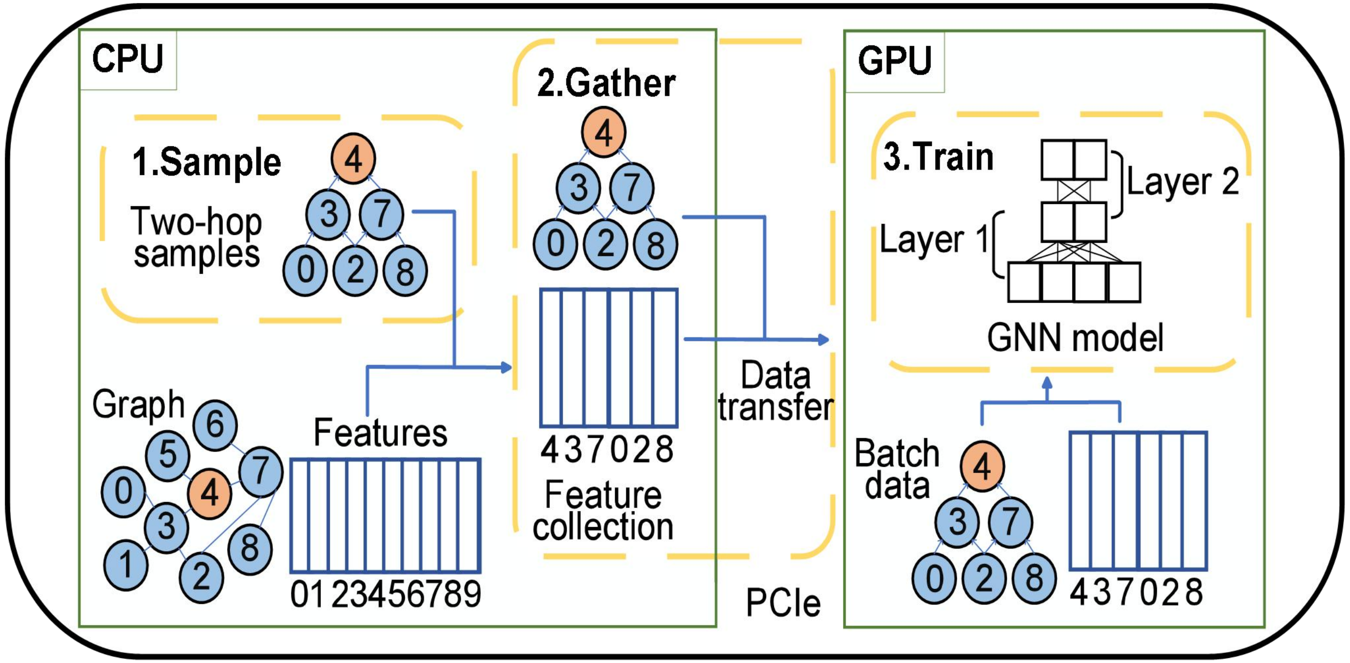
Assigning three steps
to CPU and GPU

CPU

GPU

Existing GNN Systems

Step-based task orchestrating methods



DGL [Arxiv'19]

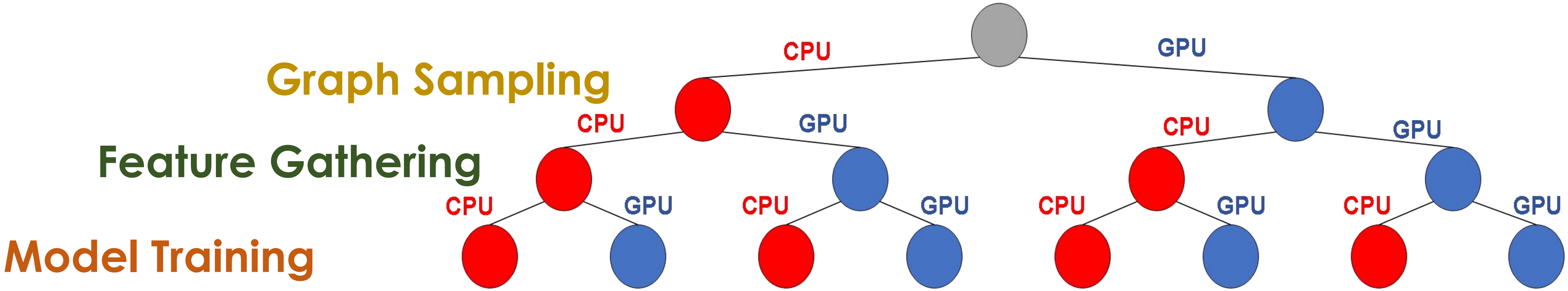
CPU

Graph Sampling
Feature Gathering

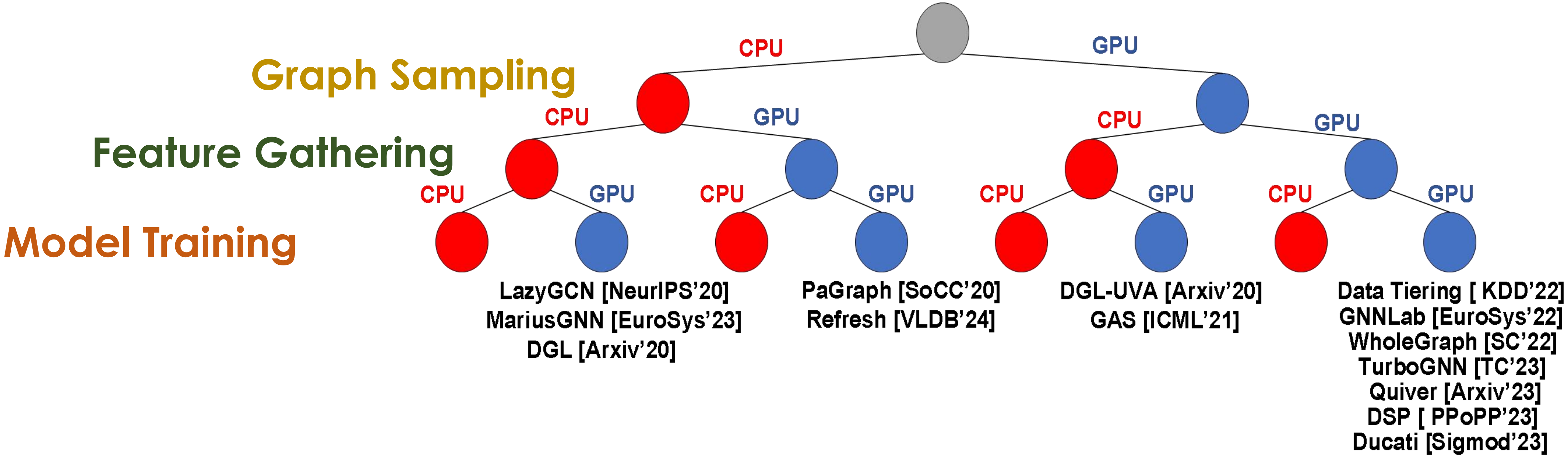
GPU

Model Training

Task Orchestrating Method Classification



Task Orchestrating Method Classification



Existing task orchestrating methods contain mainly four cases

Case 1: Placing Sample and Gather on CPUs

CPU

Graph Sampling
Feature Gathering

GPU

Model Training

Graph Sampling and Feature Gather occupy 80.5 % of the total runtime

Dataset	Sample	Gather (FC)	Gather (FT)	Total
Reddit	2.7/11%	9.1/38%	6.0/25%	23.7
Lj-large	128.8/14%	384.4/41%	252.5/27%	935.3
Orkut	78.8/10%	384.3/48%	249.1/31%	813.3
Wikipedia	209.4/12%	651.8/40%	570.9/33%	1669.1
Products	9.9/37%	7.2/27%	4.1/15%	26.8
Papers100M	11.5/32%	8.6/24%	6.4/18%	36.84

Case 1: Placing Sample and Gather on CPUs

Heavy

CPU

Graph Sampling

Feature Gathering

light

GPU

Model Training

Dataset	Sample	Gather (FC)	Gather (FT)	Total
Reddit	2.7/11%	9.1/38%	6.0/25%	23.7
Lj-large	128.8/14%	384.4/41%	252.5/27%	935.3
Orkut	78.8/10%	384.3/48%	249.1/31%	813.3
Wikipedia	209.4/12%	651.8/40%	570.9/33%	1669.1
Products	9.9/37%	7.2/27%	4.1/15%	26.8
Papers100M	11.5/32%	8.6/24%	6.4/18%	36.84

Issues:

- inefficient CPU processing
- Low GPU utilization

Case 2: Placing Sample on GPUs

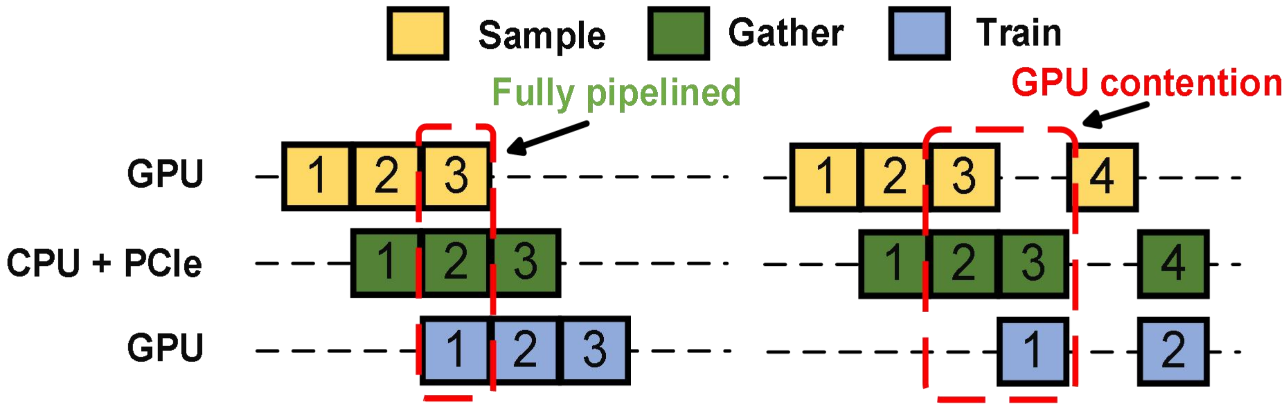
CPU

Feature Gathering

GPU

Graph Sampling

Model Training



(a) Ideal situation (b) Actual situation

Case 2: Placing Sample on GPUs

CPU

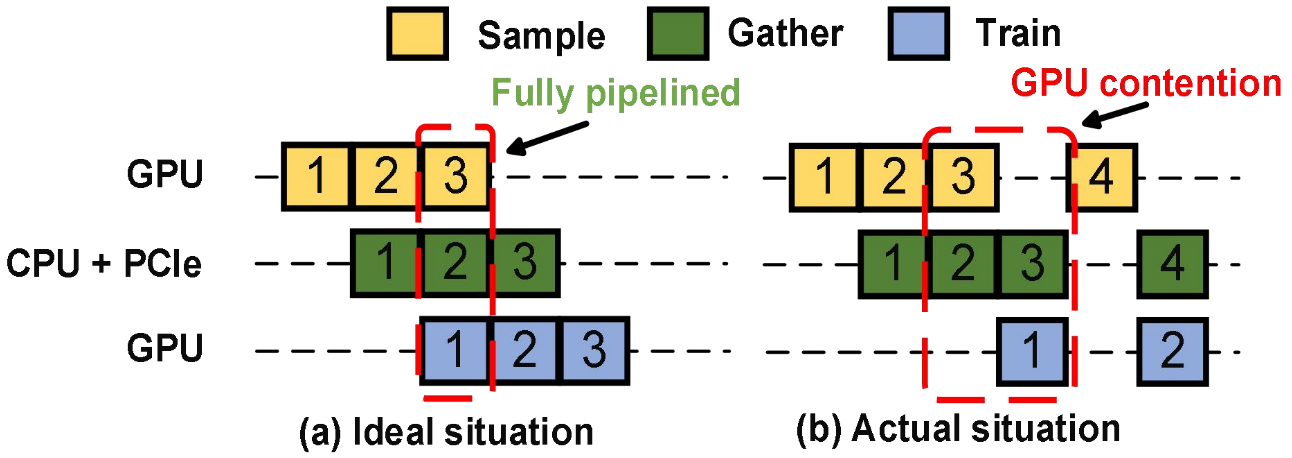
Feature Gathering

GPU

Graph Sampling

Model Training

Graph Sampling and Model Training competes for GPU computation resources



Configuration	S	G	T	Total	+pipeline
CPU-based sampling	2.28	2.84	2.76	7.88	3.42 (-56.6%)
GPU-based sampling	0.78	2.69	2.75	6.22	3.54 (-43.1%)

Case 2: Placing Sample on GPUs

CPU

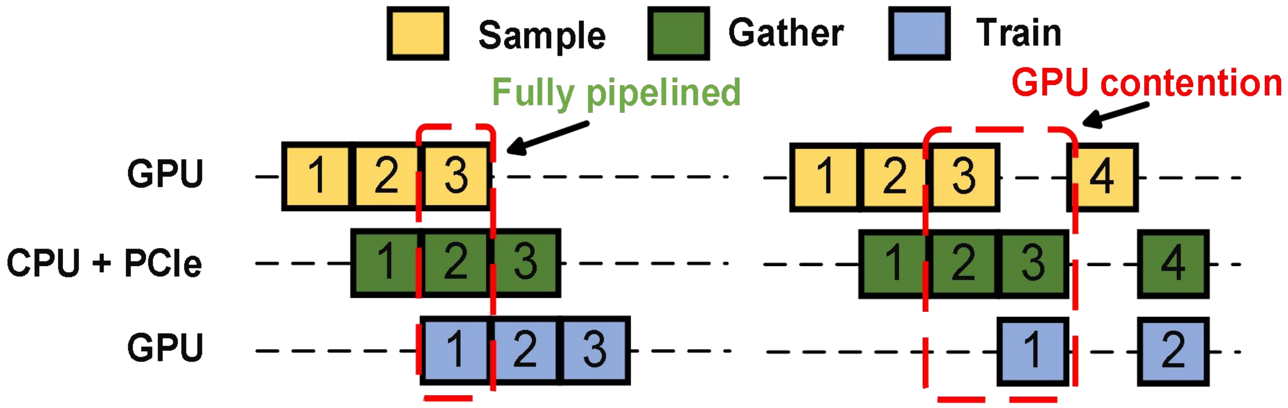
Feature Gathering

GPU

Graph Sampling

Model Training

Graph Sampling and Model Training competes for GPU computation resources



(a) Ideal situation (b) Actual situation

Configuration	S	G	T	Total	+pipeline
CPU-based sampling	2.28	2.84	2.76	7.88	3.42 (-56.6%)
GPU-based sampling	0.78	2.69	2.75	6.22	3.54 (-43.1%)

Case 2: Placing Sample on GPUs

CPU

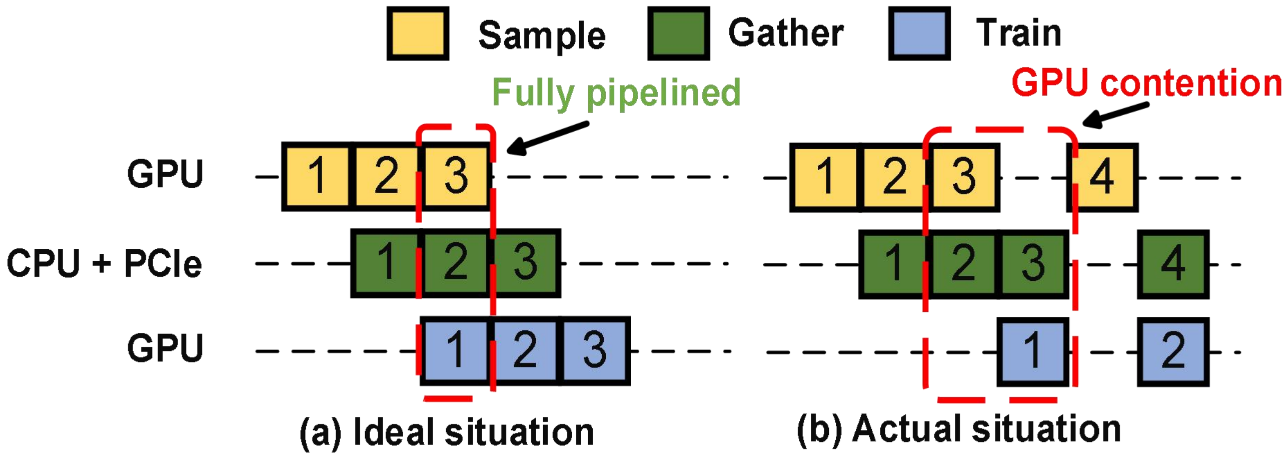
Feature Gathering

GPU

Graph Sampling

Model Training

Graph Sampling and **Model Training** competes for **GPU computation resources**



Configuration	S	G	T	Total	+pipeline
CPU-based sampling	2.28	2.84	2.76	7.88	3.42 (-56.6%)
GPU-based sampling	0.78	2.69	2.75	6.22	3.54 (-43.1%)

Case 2: Placing Sample on GPUs

CPU

light

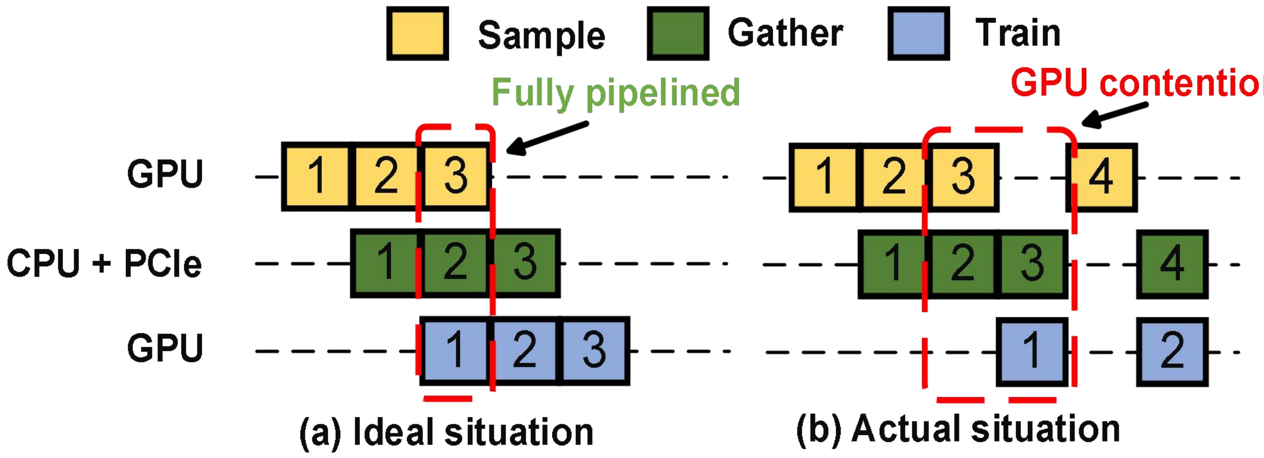
Feature Gathering

GPU

Heavy

Graph Sampling

Model Training



Issues:

- GPU resource contention

Case 3: Placing Gather on GPUs

CPU

Graph Sampling

GPU

Feature Gathering

Model Training

Feature Gather and Model Training competes for GPU memory resources

Case 3: Placing Gather on GPUs

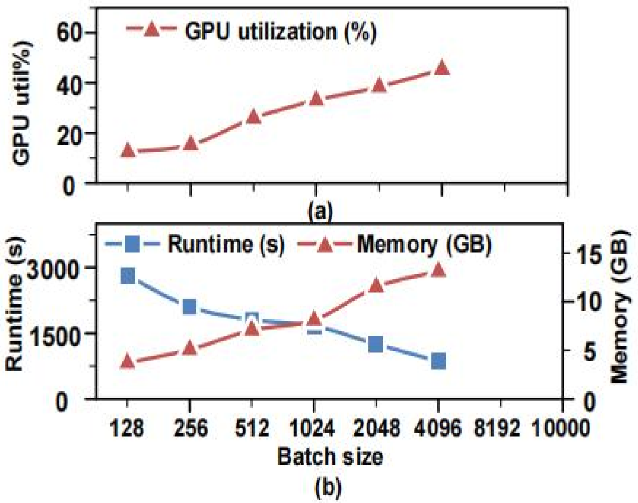
CPU

Graph Sampling

GPU

Feature Gathering

Model Training



Large batch size

- High GPU utilization
- Faster execution

Case 3: Placing Gather on GPUs

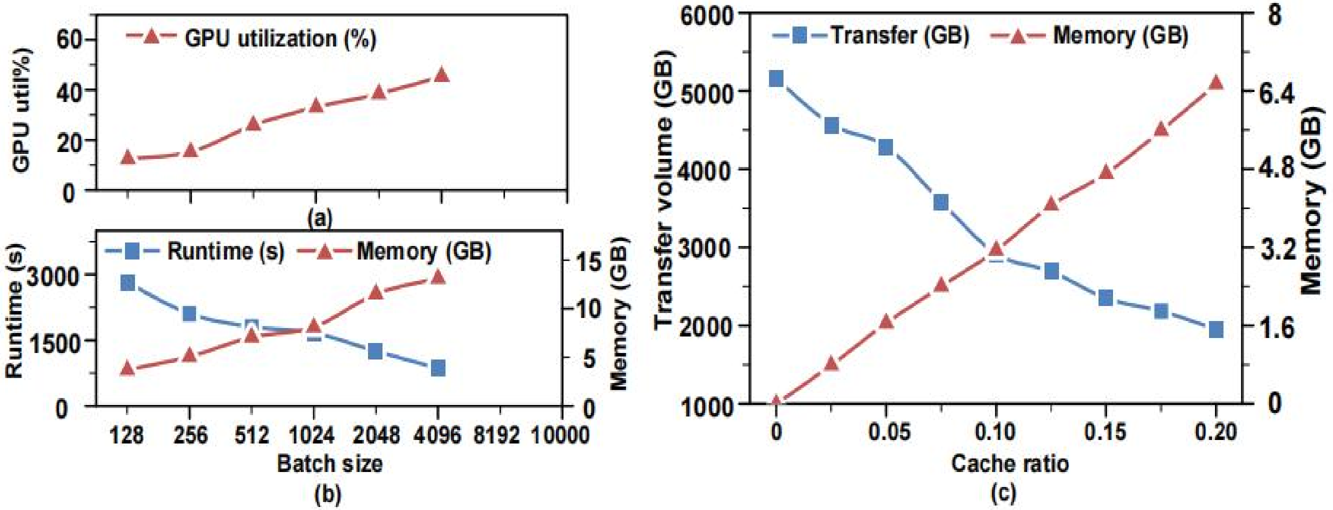
CPU

Graph Sampling

GPU

Feature Gathering

Model Training



Large batch size

- High GPU utilization
- Faster execution

High cache ratio

- Transfer reduction

Case 3: Placing Gather on GPUs

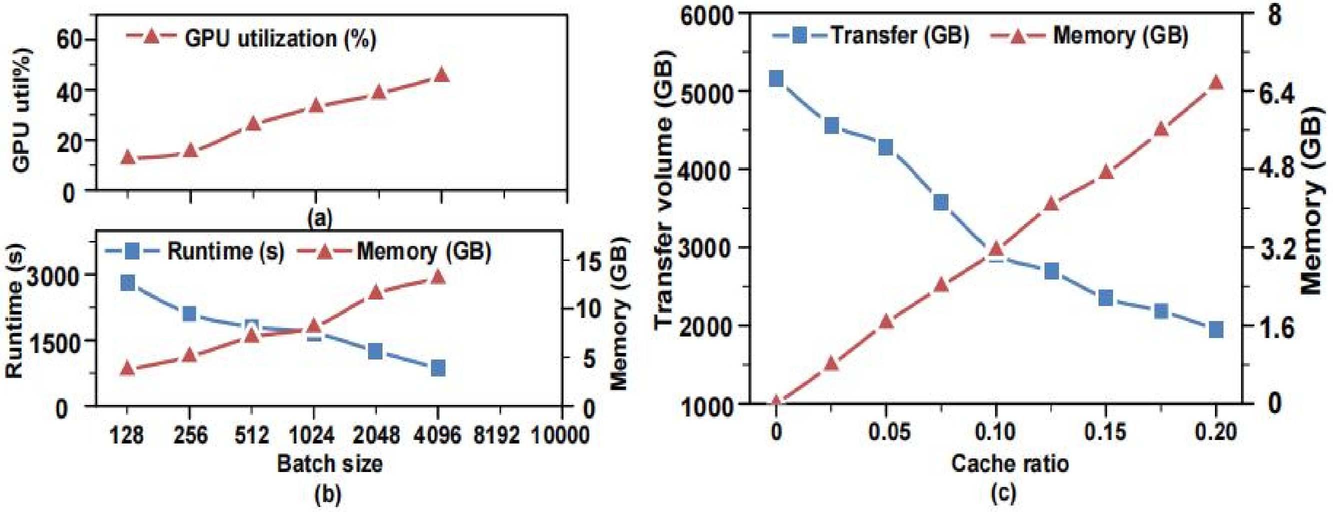
CPU

Graph Sampling

GPU

Feature Gathering

Model Training



Large batch size

High cache ratio

hard to get both

cache ratio 0.05 0.37

batch size 4096 128

Case 3: Placing Gather on GPUs

Feature Gather and **Model Training** competes for **GPU memory resources**

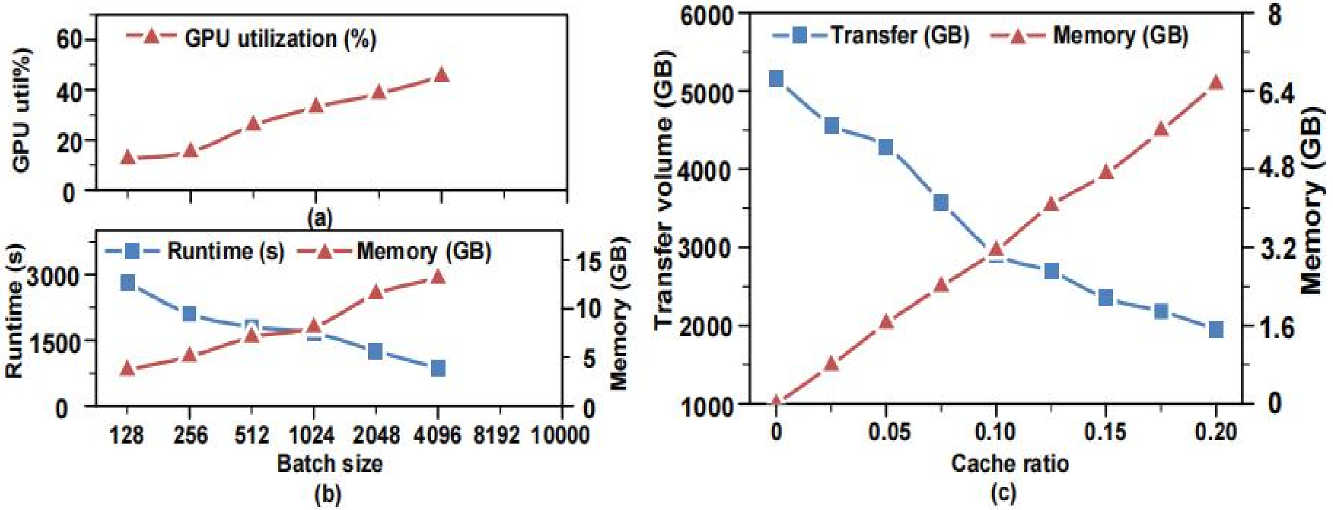
CPU

Graph Sampling

GPU

Feature Gathering

Model Training



Issues:

- GPU memory contention

Case 4: Placing Sample and Gather on GPUs

CPU

GPU

Graph Sampling

Feature Gathering

Model Training

Placing **all three steps** on the GPU suffers from **GPU memory and resource contention** in case 3 and case 4

Case 4: Placing Sample and Gather on GPUs

- **Case 4:**

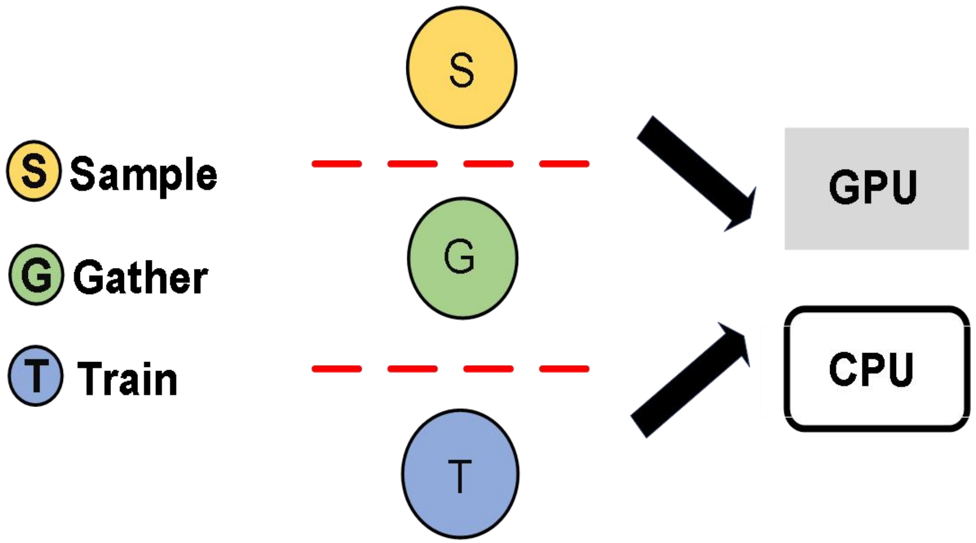
	CPU	
light		
	GPU	Graph Sampling Feature Gathering Model Training
Heavy		

Placing all three steps on the GPU suffers from GPU memory and resource contention in case 3 and case 4

Issues:
- GPU memory and resource contention
- CPU idle

Summary

Step-based task orchestrating leads to an imbalanced allocation of computational and memory resources



3 steps to 2 devices

imbalanced

Issue 1 GPU resource contention

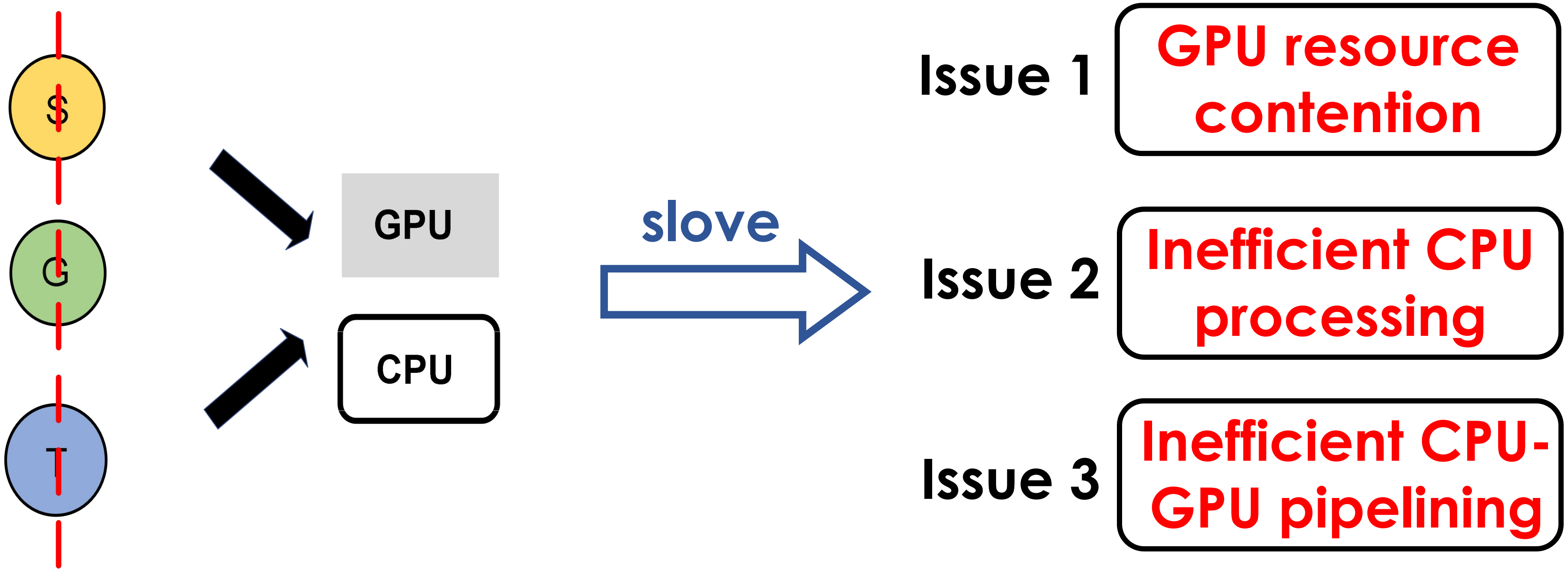
Issue 2 Inefficient CPU processing

Issue 3 Inefficient CPU-GPU pipelining

NeutronOrch

Goal:

- **Design a new task orchestrating method that avoids dividing takes by step and fully utilizes heteogeneous resources**



new task orchestrating method ?

NeutronOrch

Contributions:

1：Hotness-aware layer-based task Orchestrating

2：Super-batch pipelined training

Issue 1 — GPU resource contention

Issue 2 — Inefficient CPU processing

Issue 3 — Inefficient CPU-GPU pipelining

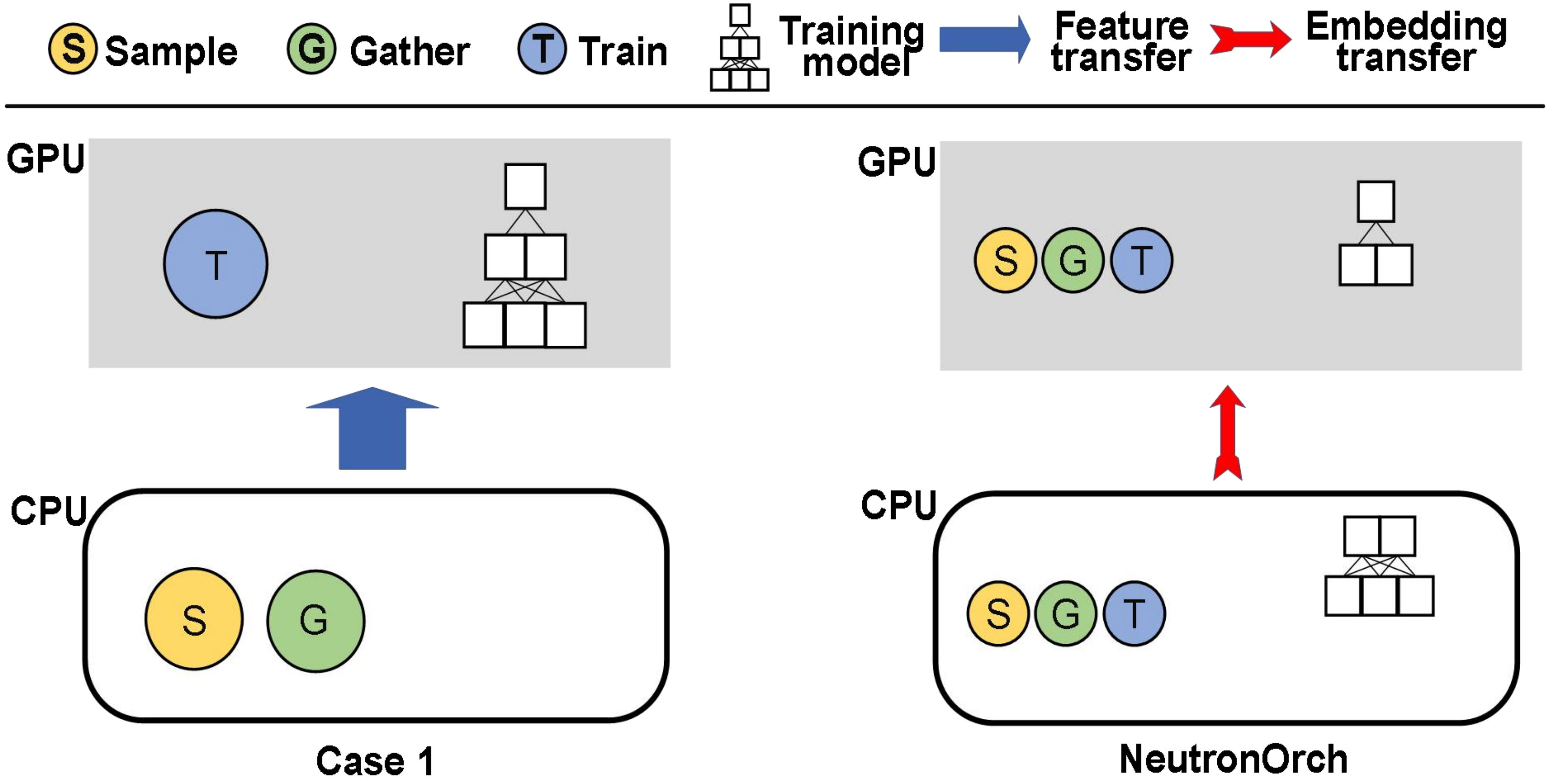
Layer-based Task Orchestrating

slove

Issue 1

GPU resource contention

We decouple the training task by layers and employ the computation of each sub-task (sample-gather-train) to a specific device



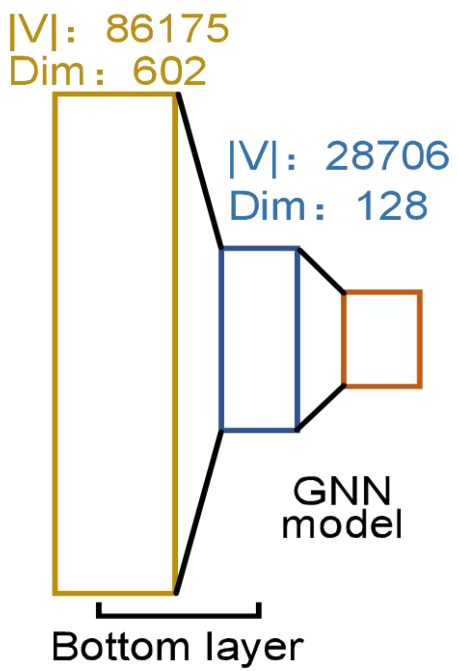
Layer-based Task Orchestrating

Offload bottom layer to CPU based on two observations:

Layer-based Task Orchestrating

Offload bottom layer to CPU based on two observations:

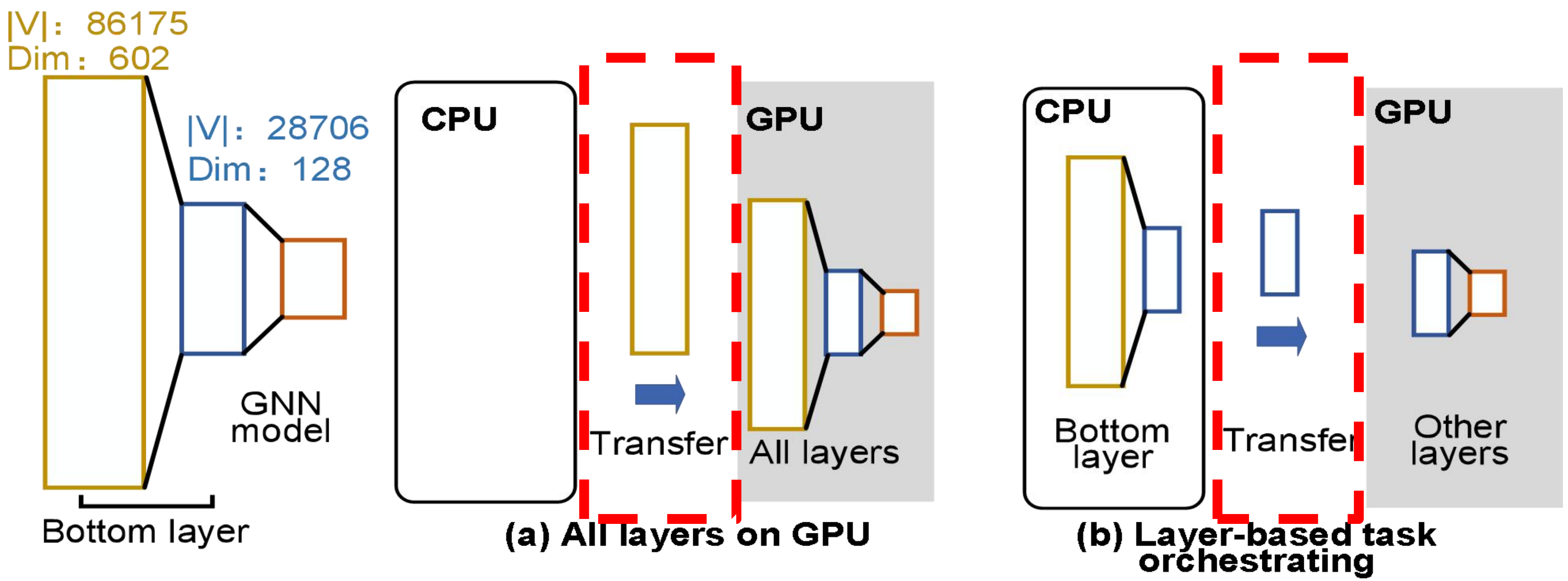
- vertices grows exponentially across layers and bottom layer constitutes over 50% of the training workload



Layer-based Task Orchestrating

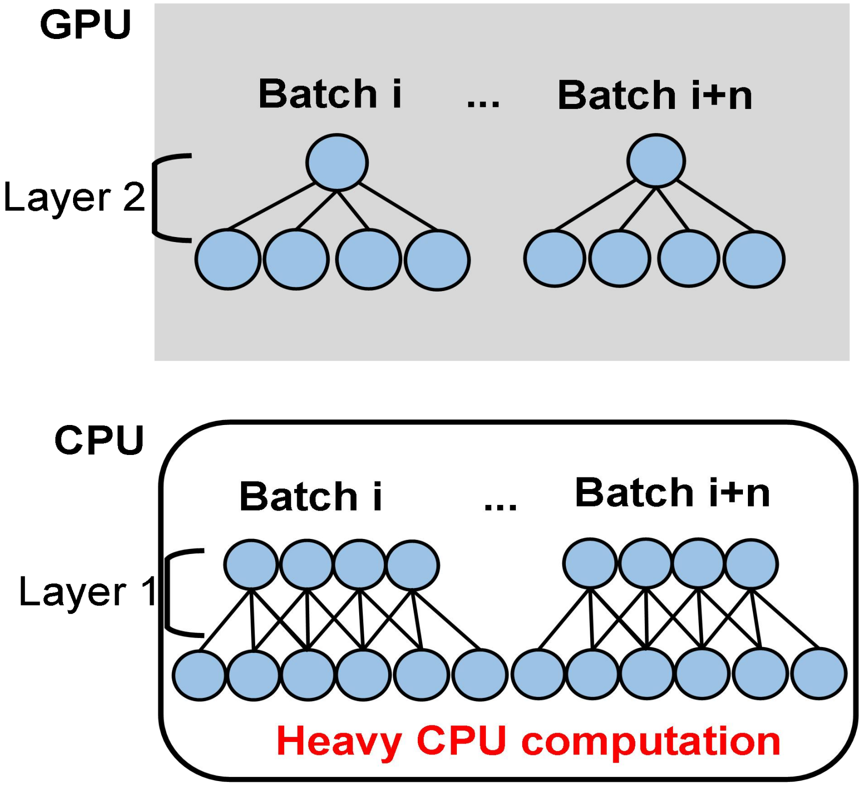
Offload bottom layer to CPU based on two observations:

- vertices grows exponentially across layers and bottom layer constitutes over 50% of the training workload
- CPU-GPU transfer overhead decreases as transferring computed embeddings instead of raw features



Layer-based Task Orchestrating

☹ Executing a **complete bottom layer** in the CPU may cause the **CPU processing a new bottleneck**

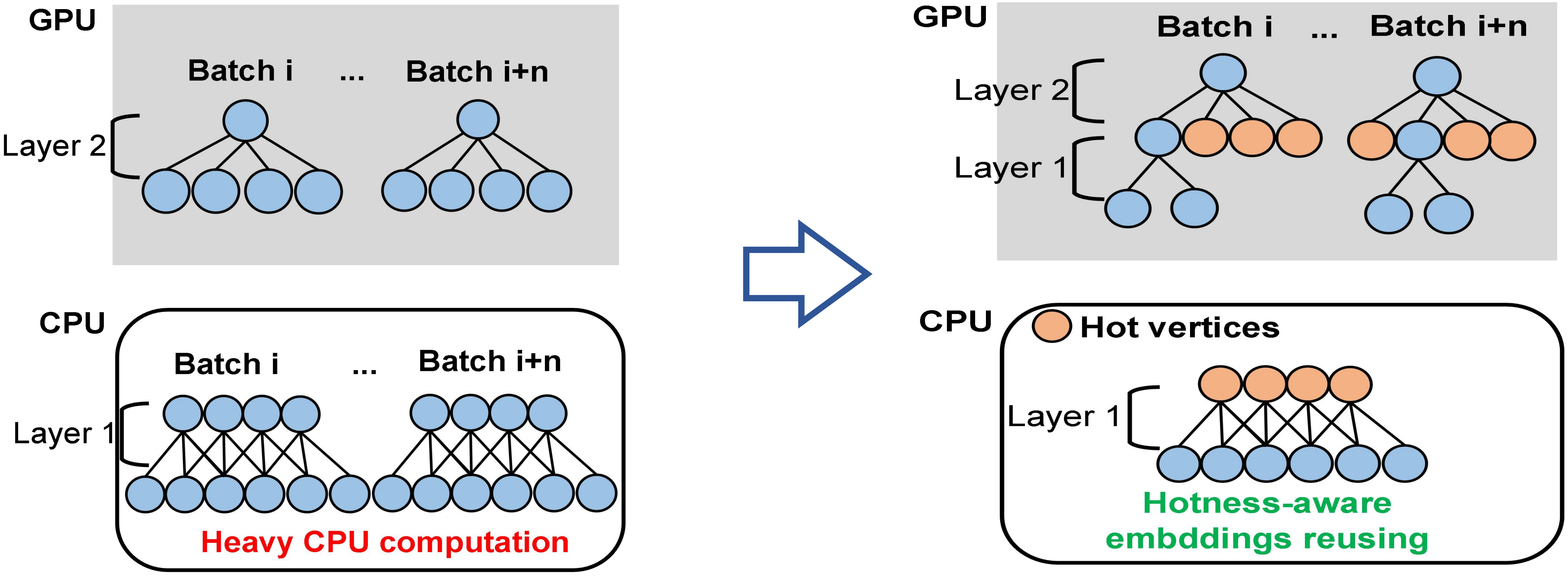


(a) naïve layer-based task orchestrating

Hotness-aware Embedding Reusing

slove → Issue 2: Inefficient CPU processing

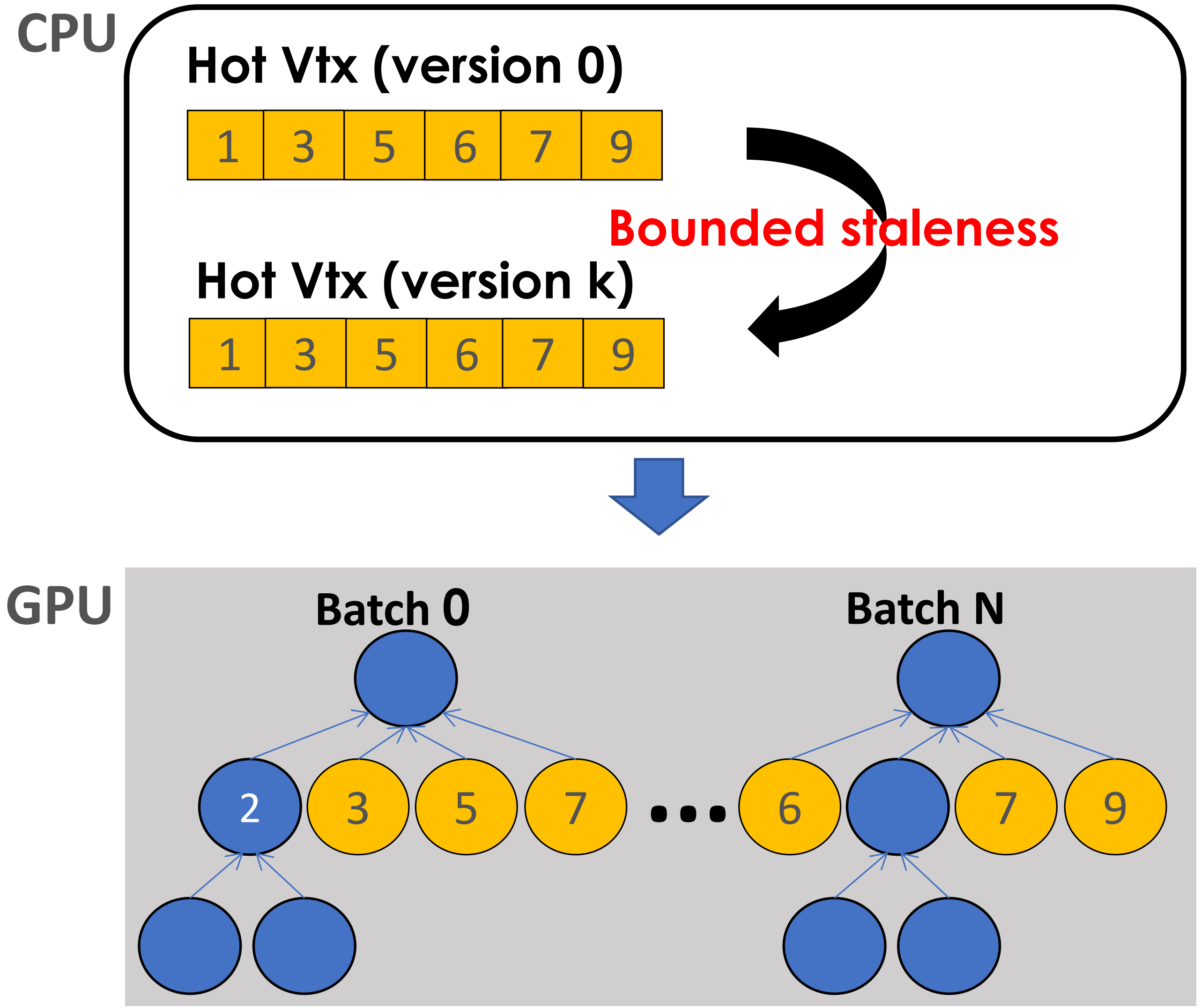
Selectively compute the embedding of frequently accessed vertices and reusing them across batches



(a) naïve layer-based task orchestrating

(b) Hotness-aware layer-based task orchestrating

Hotness-aware Embedding Reusing



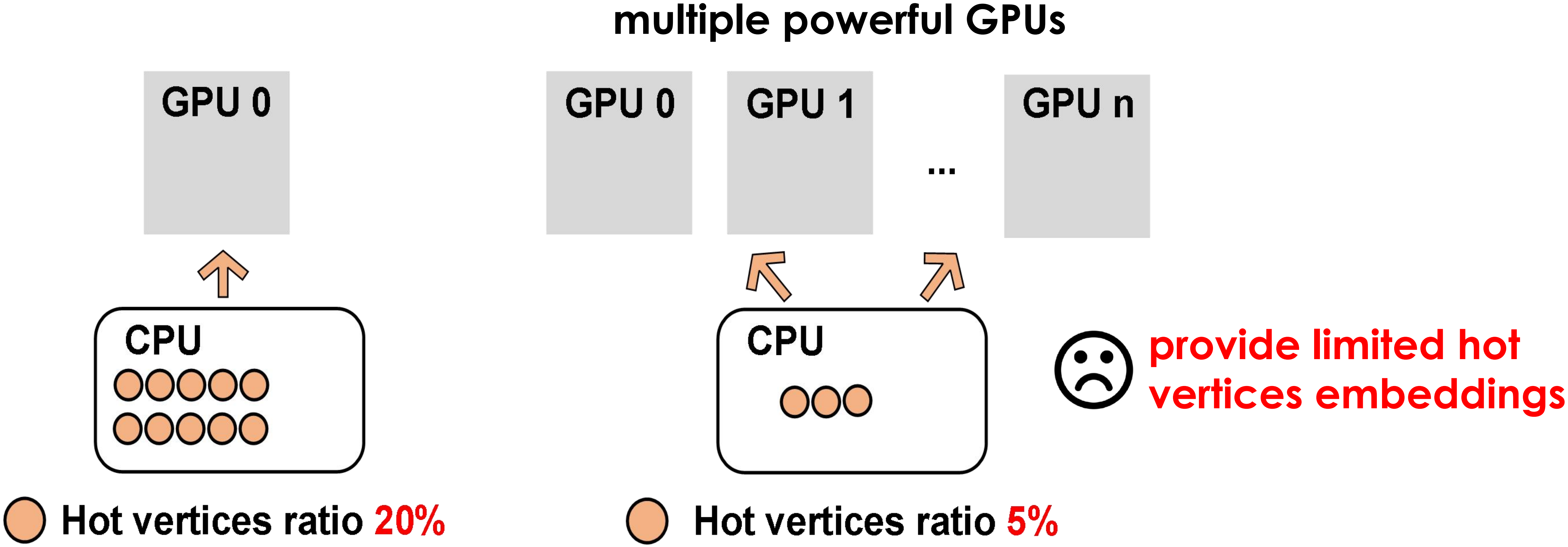
Full bottom layer embeddings for each batch

Reduce CPU computation

Hot vertice embeddings with bounded staleness

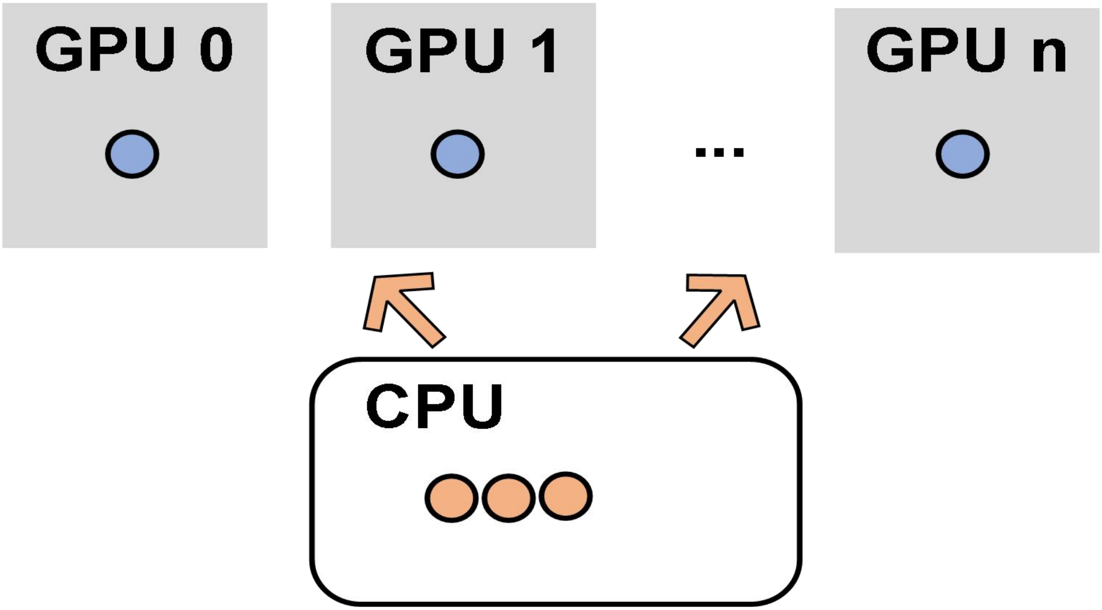
Hybrid Hot Vertices Processing

When GPU resources are significantly powerful than CPU resources, CPU computation can only provide limited contribution



Hybrid Hot Vertices Processing

Assigning hot vertices to both CPU computation and GPU feature caching



- **Minimizing the communication and computation overhead of frequently accessed vertices**
- **Maximizing the utilization of GPU and CPU memory**

Hot vertices to CPU computation 5%

Hot vertices to GPU feature cache 15%

Super-batch Pipelined Training

slove

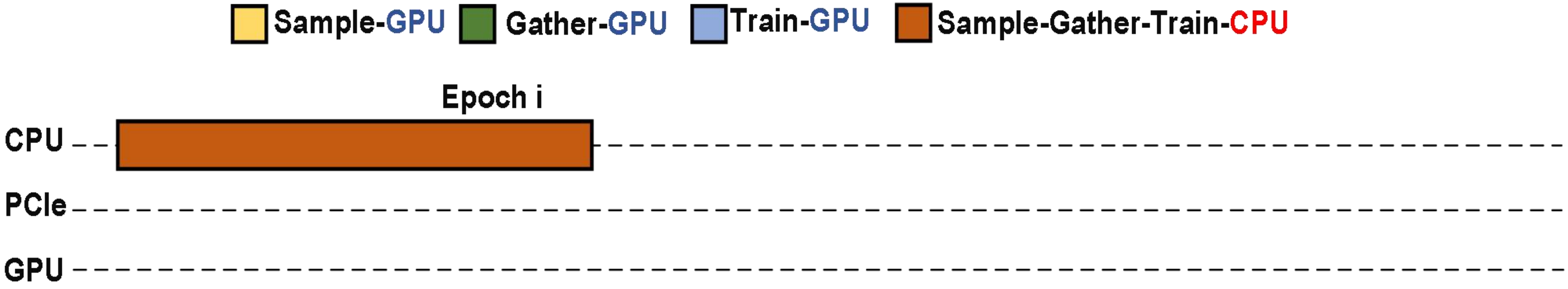
Issue 3

Inefficient CPU-GPU pipelining

Overlapping tasks across diverse computing resources is essential to achieve high performance on heterogeneous systems

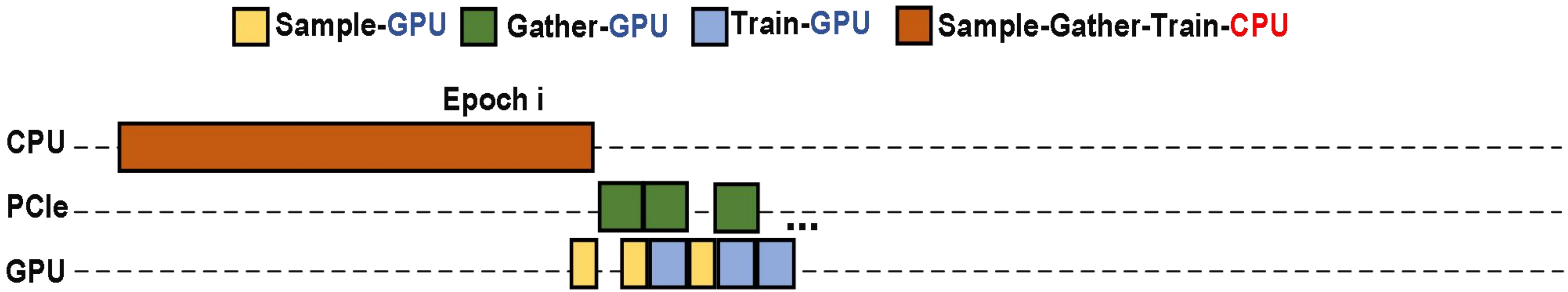
Super-batch Pipelined Training

Overlapping tasks across diverse computing resources is essential to achieve high performance on heterogeneous systems



Super-batch Pipelined Training

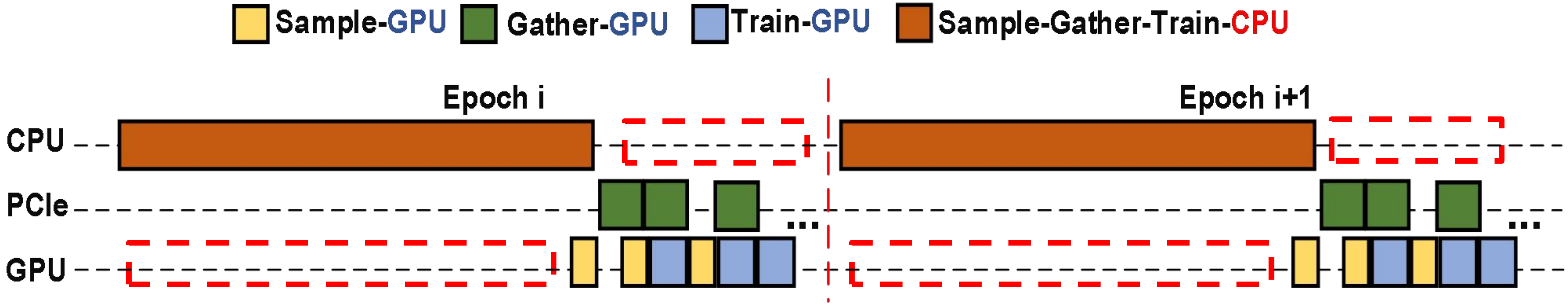
Overlapping tasks across diverse computing resources is essential to achieve high performance on heterogeneous systems



GPU training must wait for the CPU to finish the embedding computation for hot vertices

Super-batch Pipelined Training

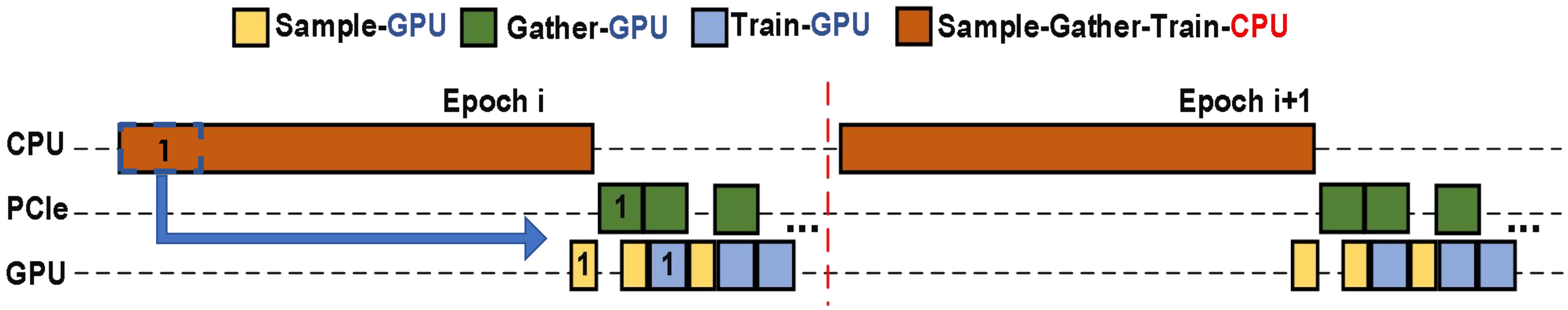
Overlapping tasks across diverse computing resources is essential to achieve high performance on heterogeneous systems



Lots of bubbles

Super-batch Pipelined Training

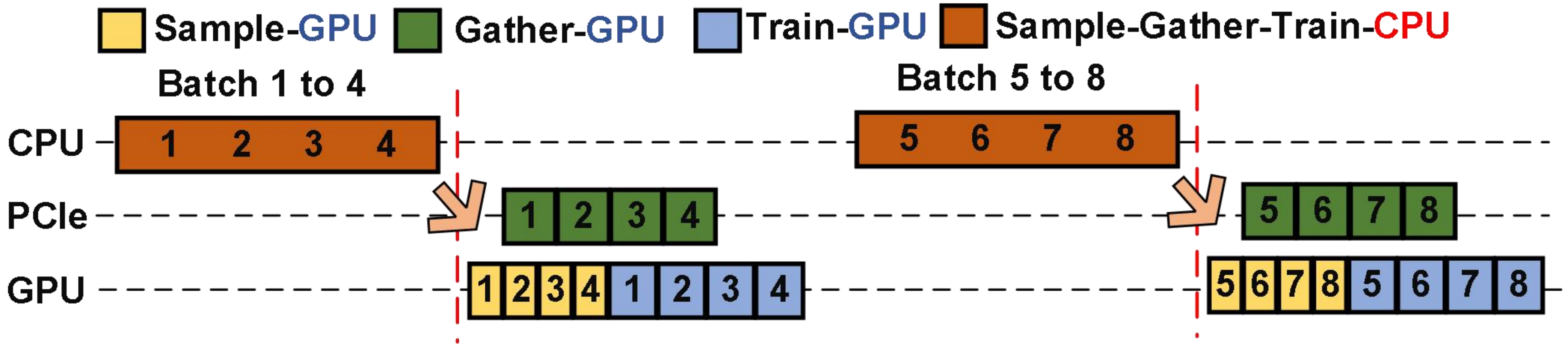
Overlapping tasks across diverse computing resources is essential to achieve high performance on heterogeneous systems



If the hot vertice embeddings required for Batch 1 are ready, GPU trianing for Batch 1 can be started earlier

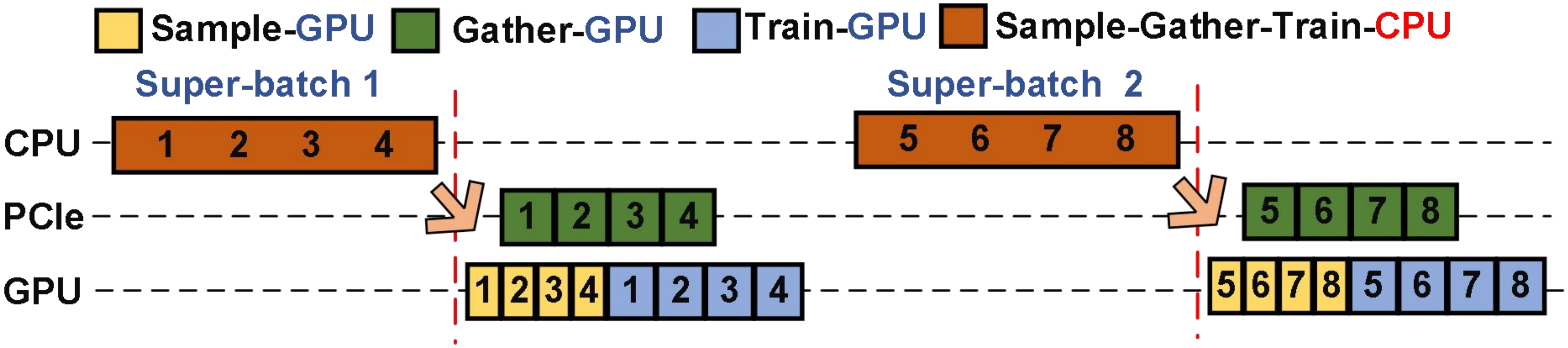
Super-batch Pipelined Training

We partition CPU computation within each epoch into multiple sub-tasks to explore pipelining opportunities



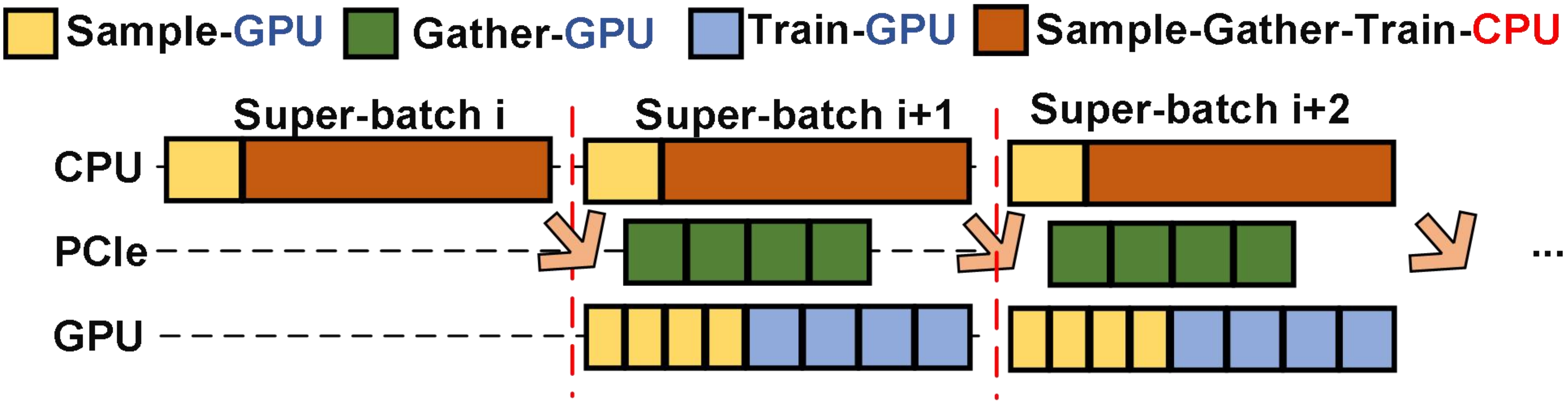
Super-batch Pipelined Training

We partition CPU computation within each epoch into multiple sub-tasks to explore pipelining opportunities



Super-batch Pipelined Training

Overlapping GPU and CPU computation tasks while strictly control the staleness of reused embeddings among super-batches



Experimental Setting

Competitors: DGL [Arxiv'20], **GNNLab** [Eurosys'22], **PaGraph** [Socc'20], **GNNAutoScale** [ICML'19], **DSP** [PPoPP'23]

Test Platforms:

Intel Xeon Platinum 8163 CPU (96 cores and 736 GB main memory) and eight NVIDIA V100 (16GB) GPUs

Algorithms and Datasets:

- 3 Graph Neural Networks
  GCN, GIN, GAT
- 6 real world graphs

Softeware Environment:

- Ubuntu 18.04 LTS
- CUDA 10.1 (418.67 driver)

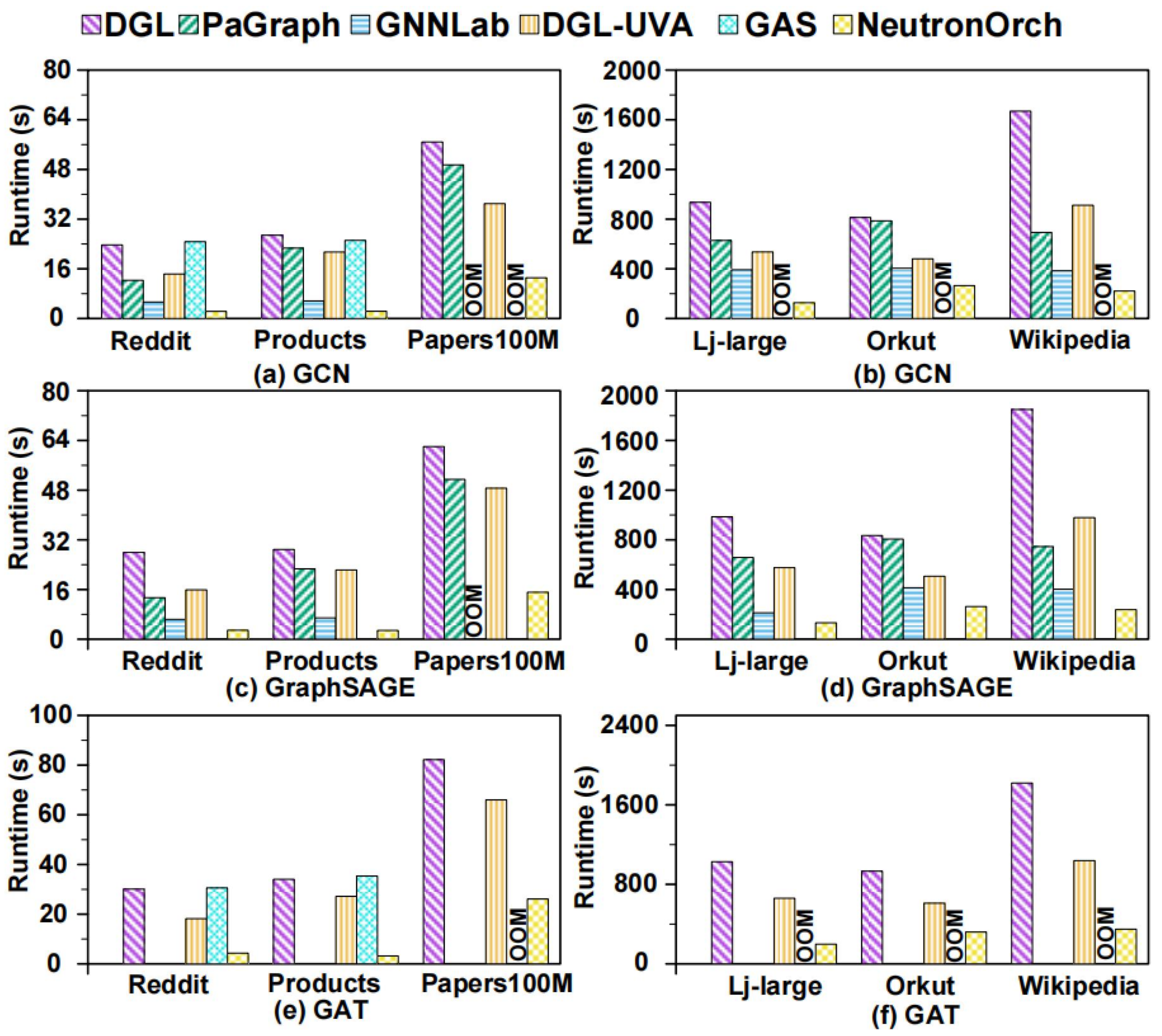
Table 4: Dataset description.

Dataset	$\|V\|$	$\|E\|$	ftr. dim	$\#\mathbb{L}$	hid. dim
Reddit [12]	232.96K	114.61M	602	41	256
Lj-large [1]	10.69M	224.61M	400	60	256
Orkut [51]	3.1M	117M	600	20	160
Wikipedia [23]	13.6M	437.2M	600	16	128
Products (PR) [14]	2.4M	61.9M	100	47	64
Papers100M (PA) [14]	111M	1.6B	128	172	64

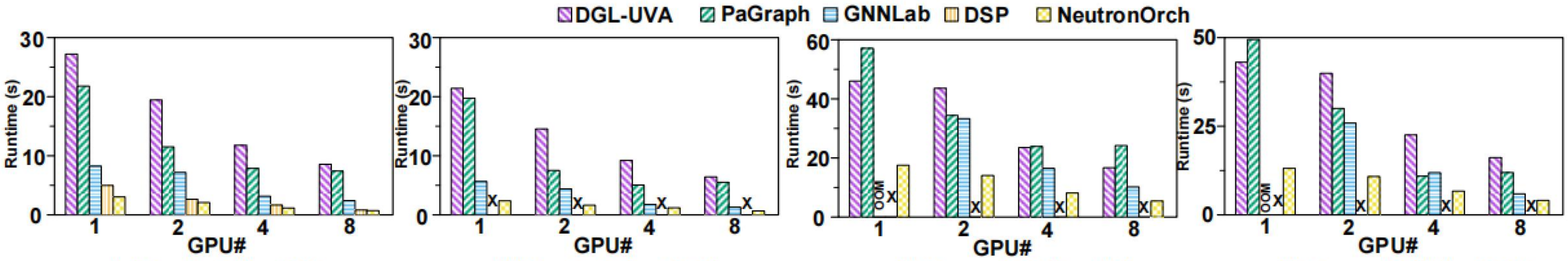
Overall Results

NeutronOrch shows better performance than the competitors

- 2.91X-11.51X faster than DGL
- 2.68X-9.72X faster than PaGraph
- 1.52X-2.43X faster than GNNLab
- 1.81-9.18X faster than DGL-UVA
- 7.08-11.05X faster than GNNAutoScale



Multi-GPU Performance

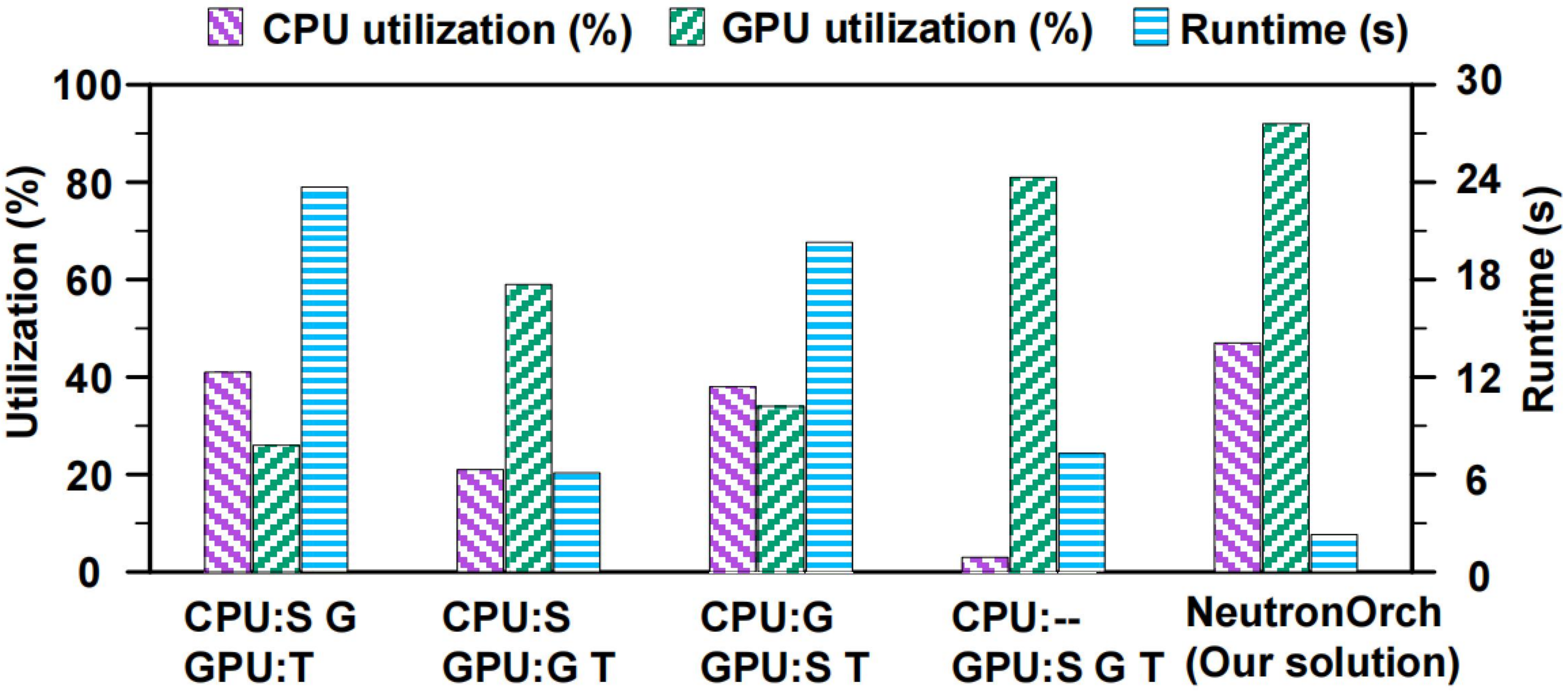


(a) Products(bs-512) (b) Products(bs-1024) (c) Papers100M(bs-512) (d) Papers100M(bs-1024)

- Compared with **DGL-UVA, PaGraph, GNNLab** and **DSP, NeutronOrch** achieves on average **6.33X, 5.20X, 2.28X,** and **1.36X** speedups
- **NeutronOrch** effectively trains large-scale GNNs by offloading computations to the CPU

CPU and GPU Utilization

S, G, and T represent the sample, gather, and train



- **NeutronOrch** fully utilizes heterogeneousresources and achieves better performance
- High GPU utilization ensures shorter runtime, while **CPU offloading boosts performance**

Summary

NeutronOrch: Rethinking Sample-based GNN Training under CPU-GPU Heterogeneous Environments

- **Providing insight into the four existing approaches**
  We provide a comprehensive analysis of resource utilization issues associated with the task orchestrating methods for sample-based GNN systems on GPU-CPU heterogeneous platforms

Summary

NeutronOrch: Rethinking Sample-based GNN Training under CPU-GPU Heterogeneous Environments

- □ **Providing insight into the four existing approaches**

  We provide a comprehensive analysis of resource utilization issues associated with the task orchestrating methods for sample-based GNN systems on GPU-CPU heterogeneous platforms

- □ **Proposing a hotness-aware layer-based task orchestrating method**

  We propose a hotness-aware layer-based task orchestrating method that effectively leverages the computation and memory resources of the GPU-CPU heterogeneous system

Summary

NeutronOrch: Rethinking Sample-based GNN Training under CPU-GPU Heterogeneous Environments

- □ **Providing insight into the four existing approaches**

  We provide a comprehensive analysis of resource utilization issues associated with the task orchestrating methods for sample-based GNN systems on GPU-CPU heterogeneous platforms

- □ **Proposing a hotness-aware layer-based task orchestrating method**

  We propose a hotness-aware layer-based task orchestrating method that effectively leverages the computation and memory resources of the GPU-CPU heterogeneous system

- □ **Proposing a super-batch pipelined task scheduling method**

  We propose a super-batch pipelined task scheduling method that seamlessly overlaps different tasks on heterogeneous resources and efficiently achieves strict bounded staleness

Summary

NeutronOrch: Rethinking Sample-based GNN Training under CPU-GPU Heterogeneous Environments

- **Providing insight into the four existing approaches**

  We provide a comprehensive analysis of resource utilization issues associated with the task orchestrating methods for sample-based GNN systems on GPU-CPU heterogeneous platforms

- **Proposing a hotness-aware layer-based task orchestrating method**

  We propose a hotness-aware layer-based task orchestrating method that effectively leverages the computation and memory resources of the GPU-CPU heterogeneous system

- **Proposing a super-batch pipelined task scheduling metho**

  We propose a super-batch pipelined task scheduling method that seamlessly o … asks on heterogeneous resources and efficiently achieves strict bounded staleness

Questions

- **The codes are publicly available on github**

  https://github.com/Aix-im/Sample-based-GNN