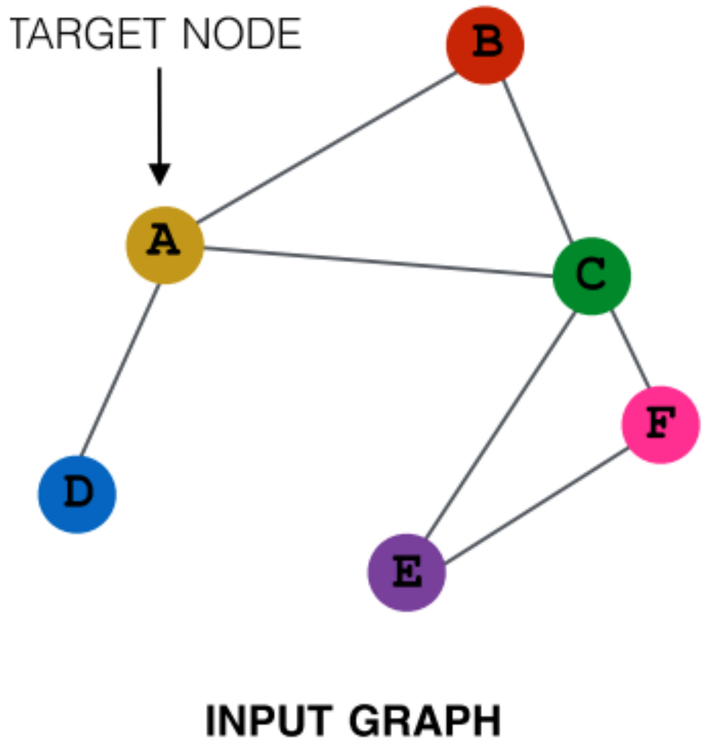
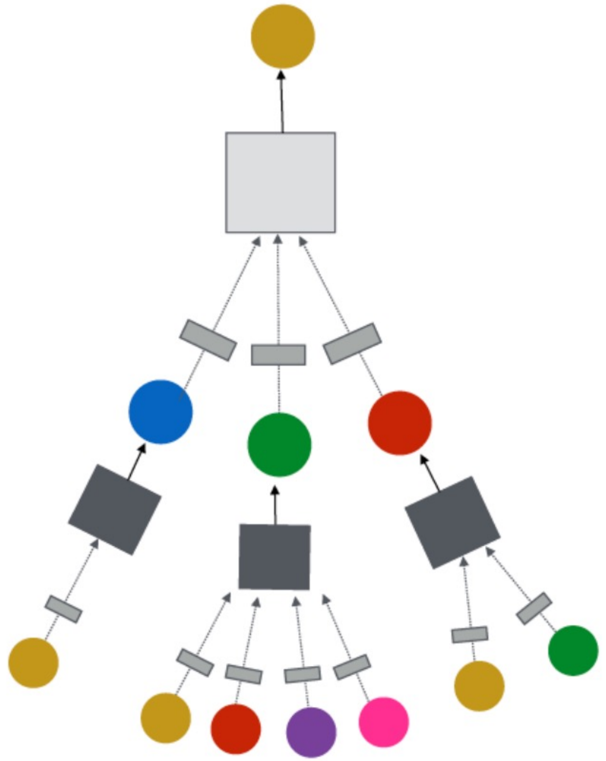
AdaQP: **Ada**ptive Message **Q**uantization and **P**arallelization for Distributed Full-graph Training

Borui Wan, Juntao Zhao, Chuan Wu

The University of Hong Kong

GNN Message Passing Paradigm





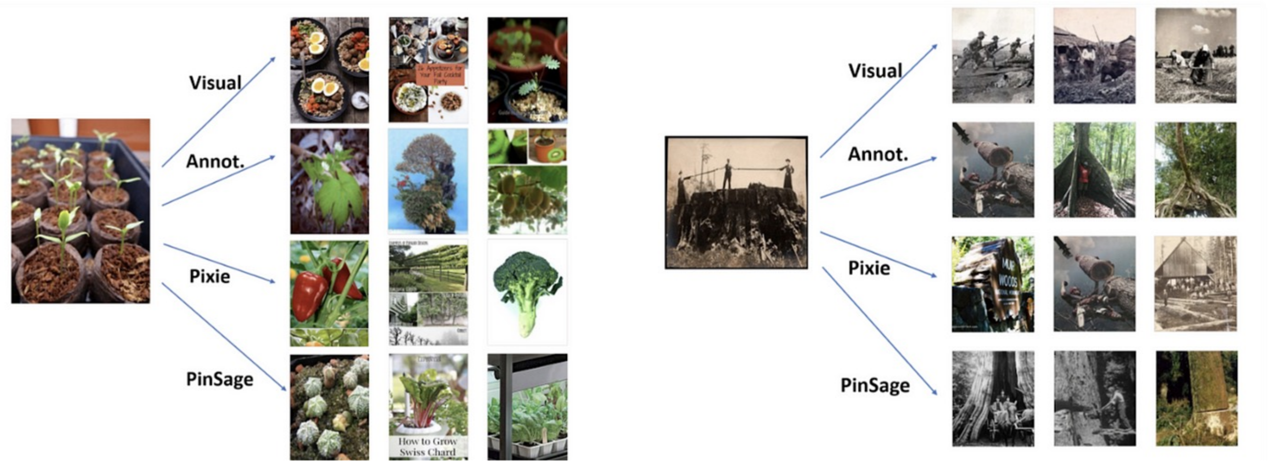
Mathematic Form

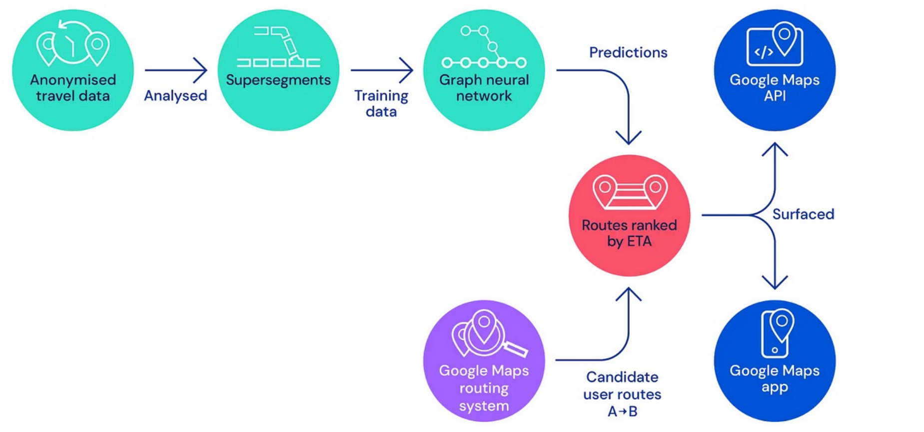
$$h_{N(v)}^{l} = \phi^{l}\left(h_u^{l-1} \middle| u \in N(v)\right)$$
$$h_v^{l} = \psi(h_v^{l-1}, h_{N(v)}^{l})$$

- h_v^l: learned embedding of node v at layer l
- $N(v)$: neighbor set of node v
- ϕ^l: aggregation function
- ψ^l: update function

Aggregate information from neighbors to produce node embeddings

Training on Large-scale Graphs



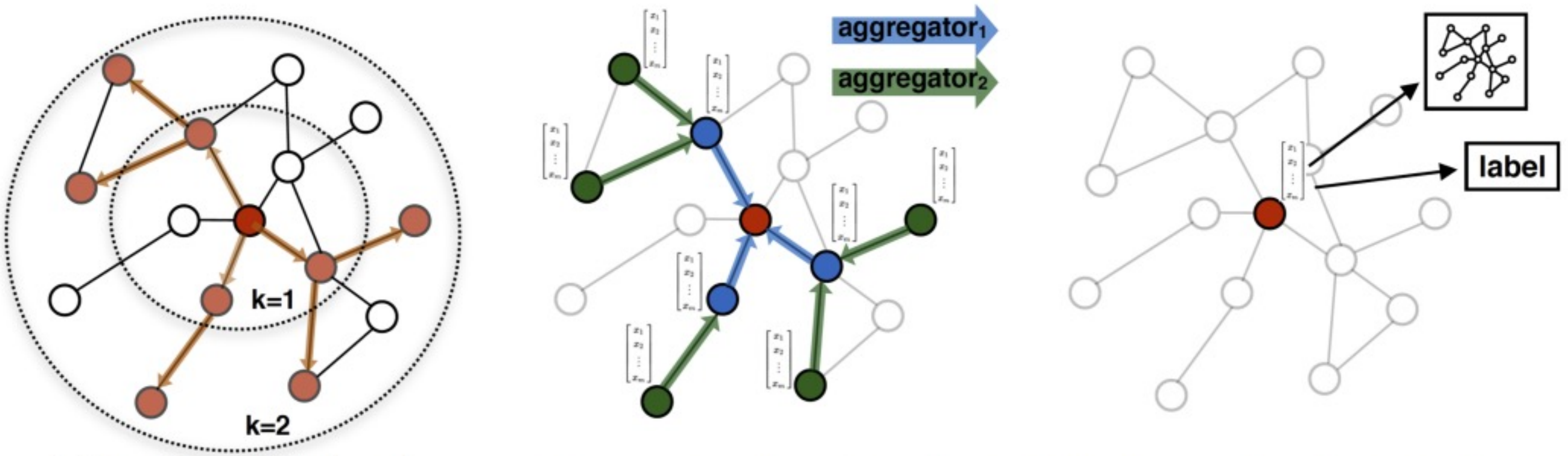


Pinterest: PinSage for web-sacle recommendation

Google Maps: GNN for traffic forecasting

Efficient training on large-scale graphs is challenging but meaningful for industry applications

Sampling-based GNN Training



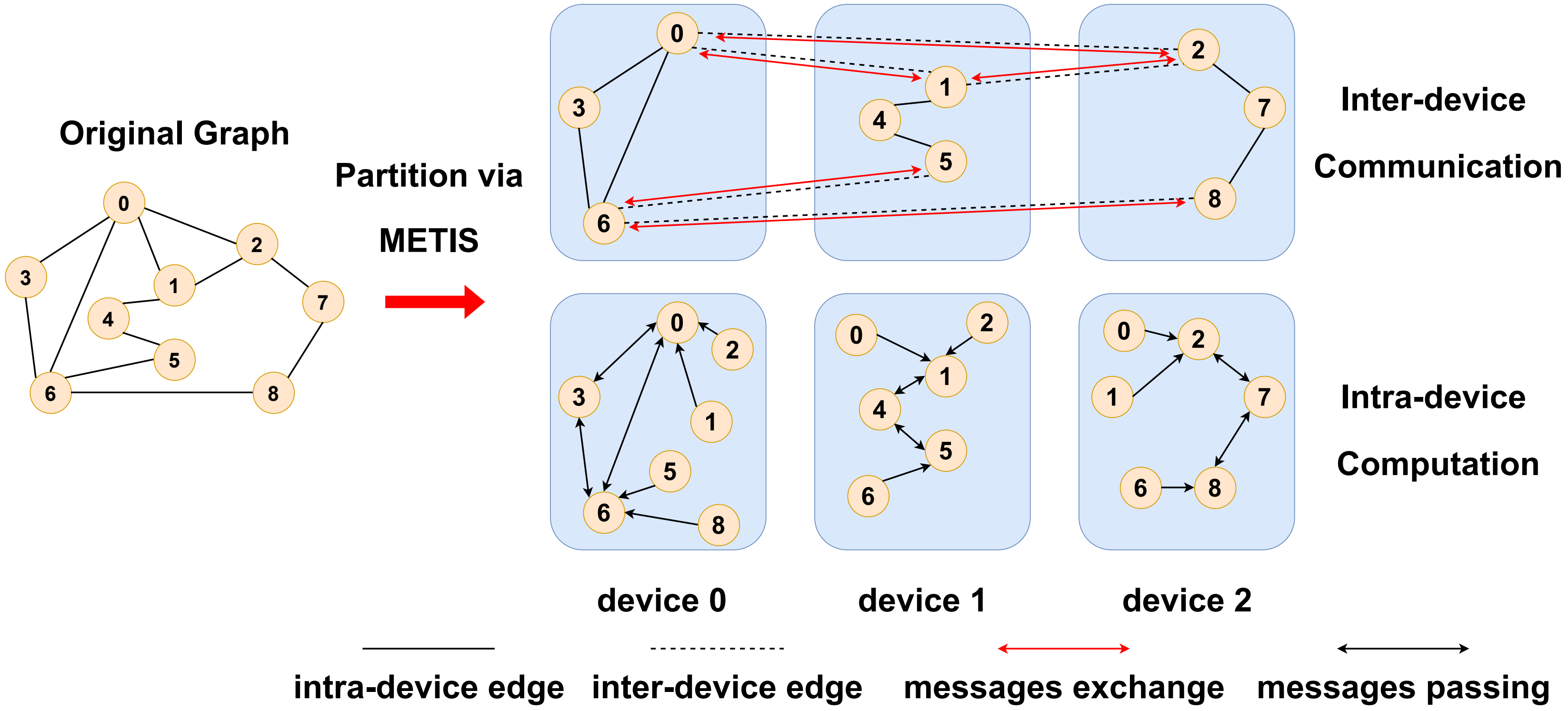
1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

GraphSAGE

Sampling-based GNN training

- Use sampled neighbors for training, altering graph topology (**lower model accuracy**)
- Transfer k-hop features for a k-layer GNN (**time-consuming**)
- Run (sophisticated) sampling algorithms (**extra overhead**)

Distributed Full-graph GNN Training



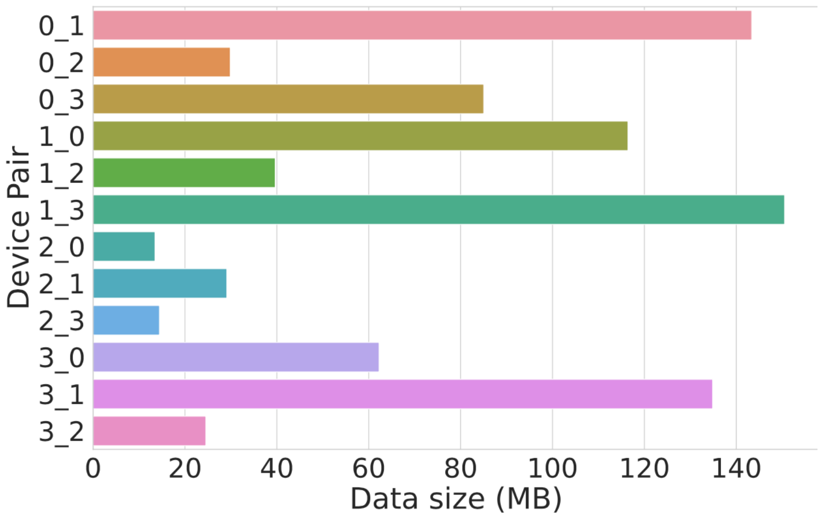
Distributed full-graph training

- Each worker holds one graph partition and remote 1-hop HALO node indices in GPU
- Large data transfer across devices due to exchanging remote messages **[features and embeddings in forward, embedding gradients in backward]** for computation in each GNN layer

Communication Bottleneck

Dataset	Partition Setting	Communication Cost	Remote Neighbor Ratio
Reddit (Hamilton et al., 2017a)	2M-1D	66.78%	41.54%
	2M-2D	75.20%	62.60%
ogbn-products (Hu et al., 2020)	2M-2D	75.59%	31.09%
	2M-4D	76.67%	40.52%
AmazonProducts (Zeng et al., 2020)	2M-2D	75.58%	39.75%
	2M-4D	78.22%	53.00%

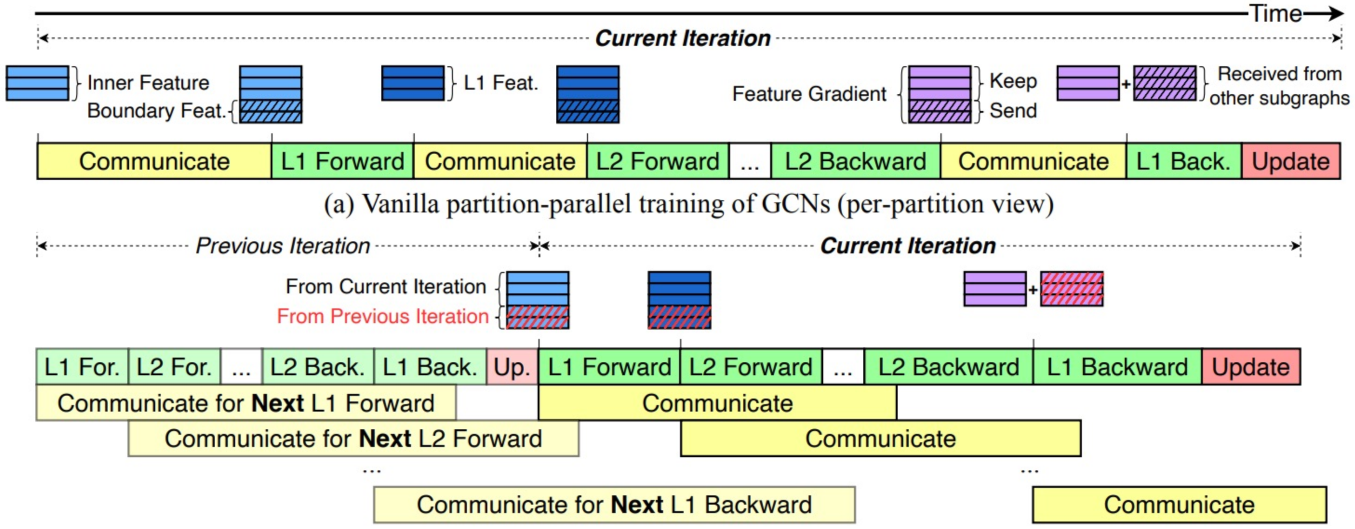
M – Machines **D – Devices per Machine**

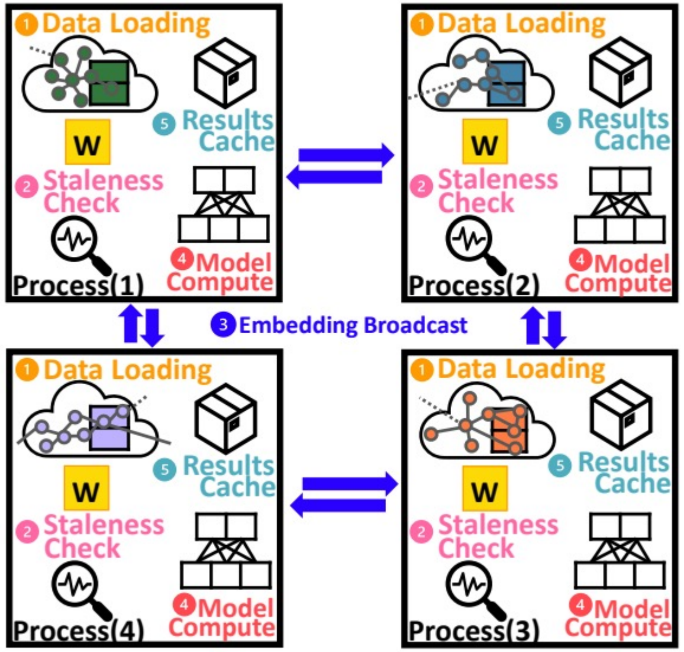


- Communication cost (dividing average communication time by average per-epoch training time among all devices) can be **up to 78.22%**
- As parallelism level increases, communication overhead becomes more severe
- Unbalanced number of messages transferred across difference device pairs

Careful design to alleviate such graph data transfer overhead is the key for training acceleration

SOTA Works





PipeGCN

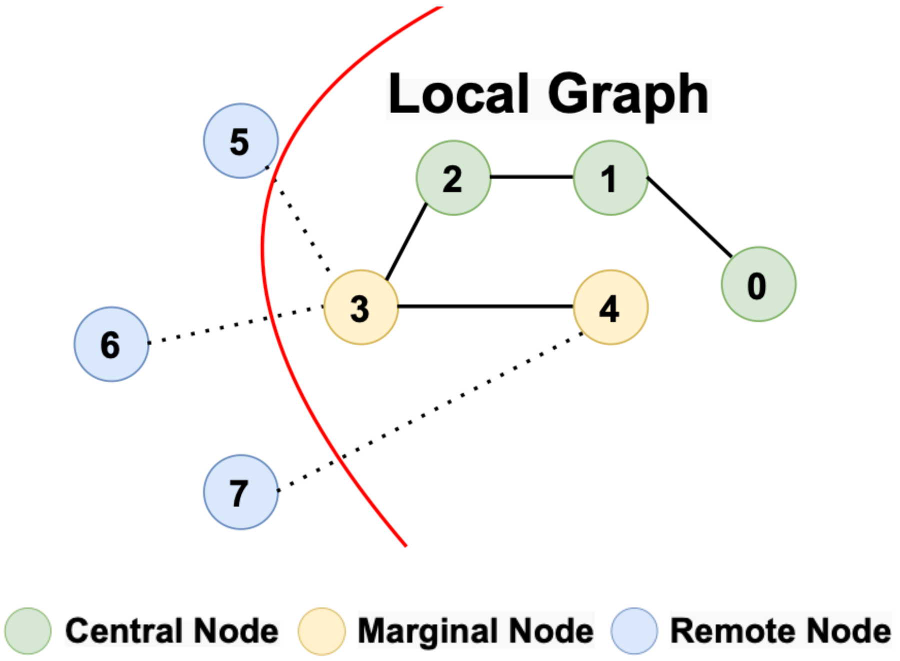
- hide communication overhead by pipelining computation with communication across epochs

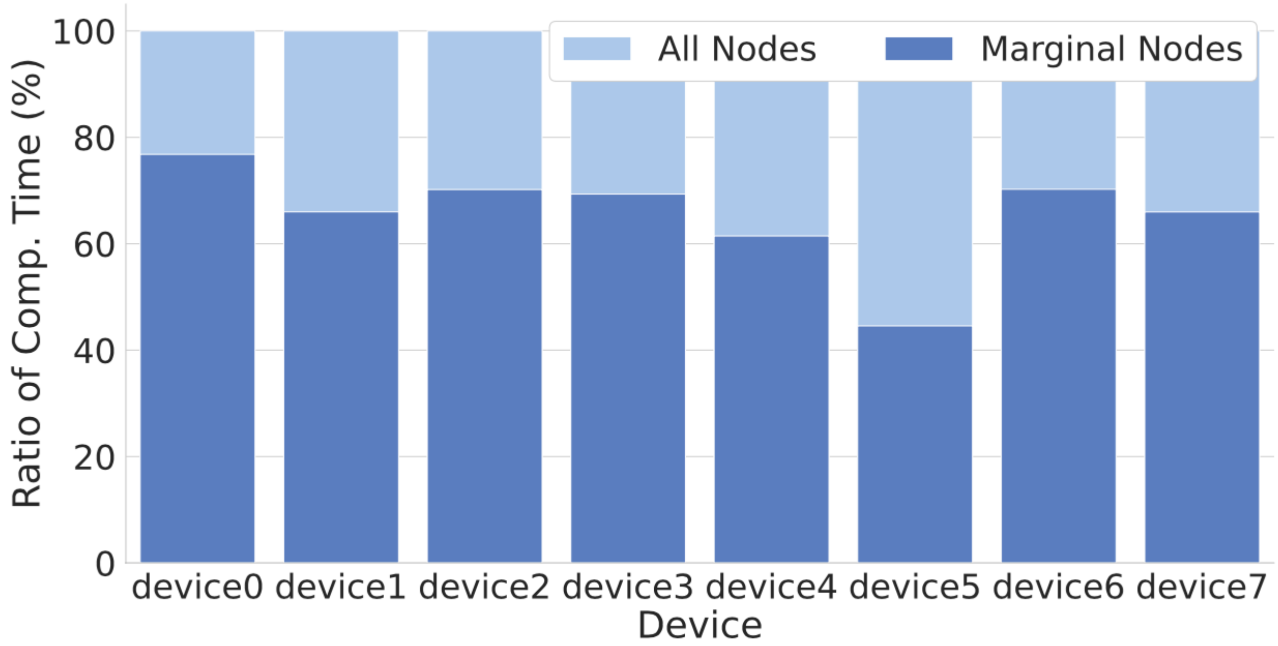
SANCUS

- Check embedding staleness before broadcast in each epoch
- Reuse stale embedding from Results Cache

SOTA works adopt staleness-based communication-hiding design, which hurts training convergence

Opportunity for Hiding Computation Time

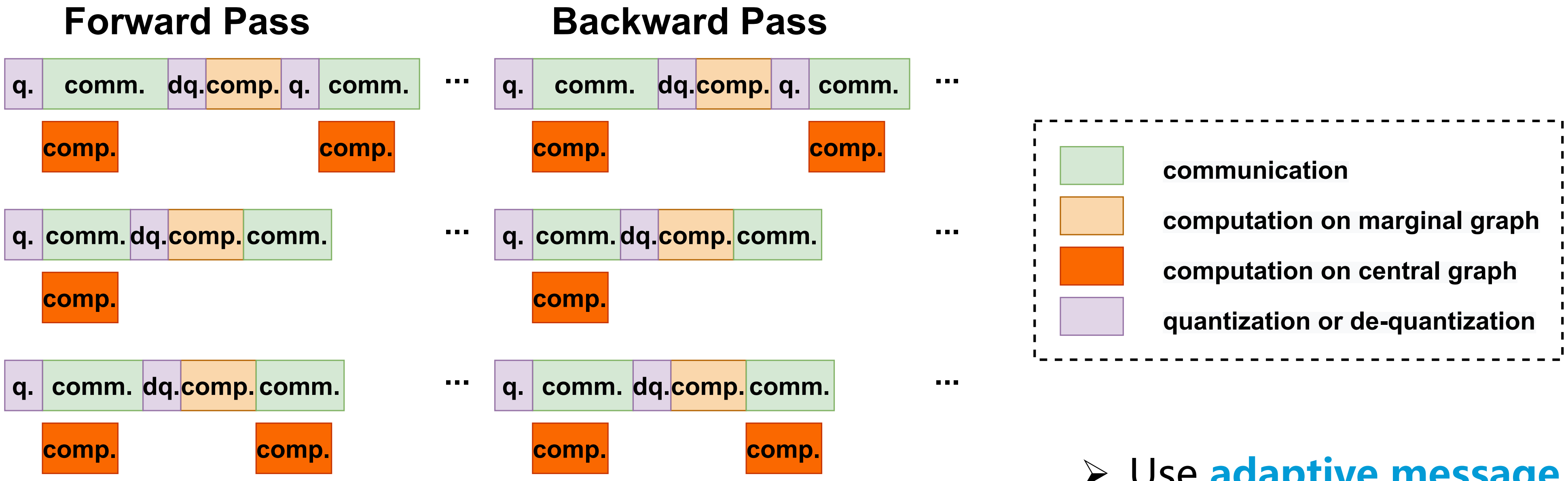




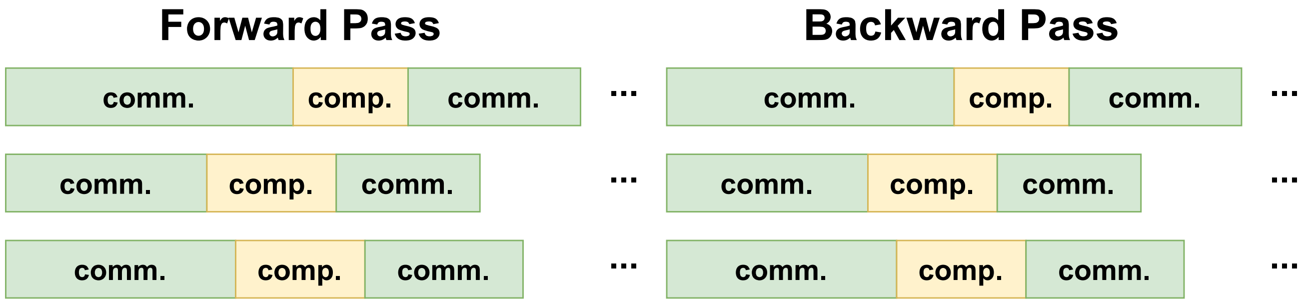
Training GCN on ogbn-products with 8 partitions

- Divide local nodes into ***central nodes*** (without remote neighbors) and ***marginal nodes*** (with remote neighbors)
- Central nodes computation can be hidden within marginal nodes communication

Parallelization + Message Quantization Design



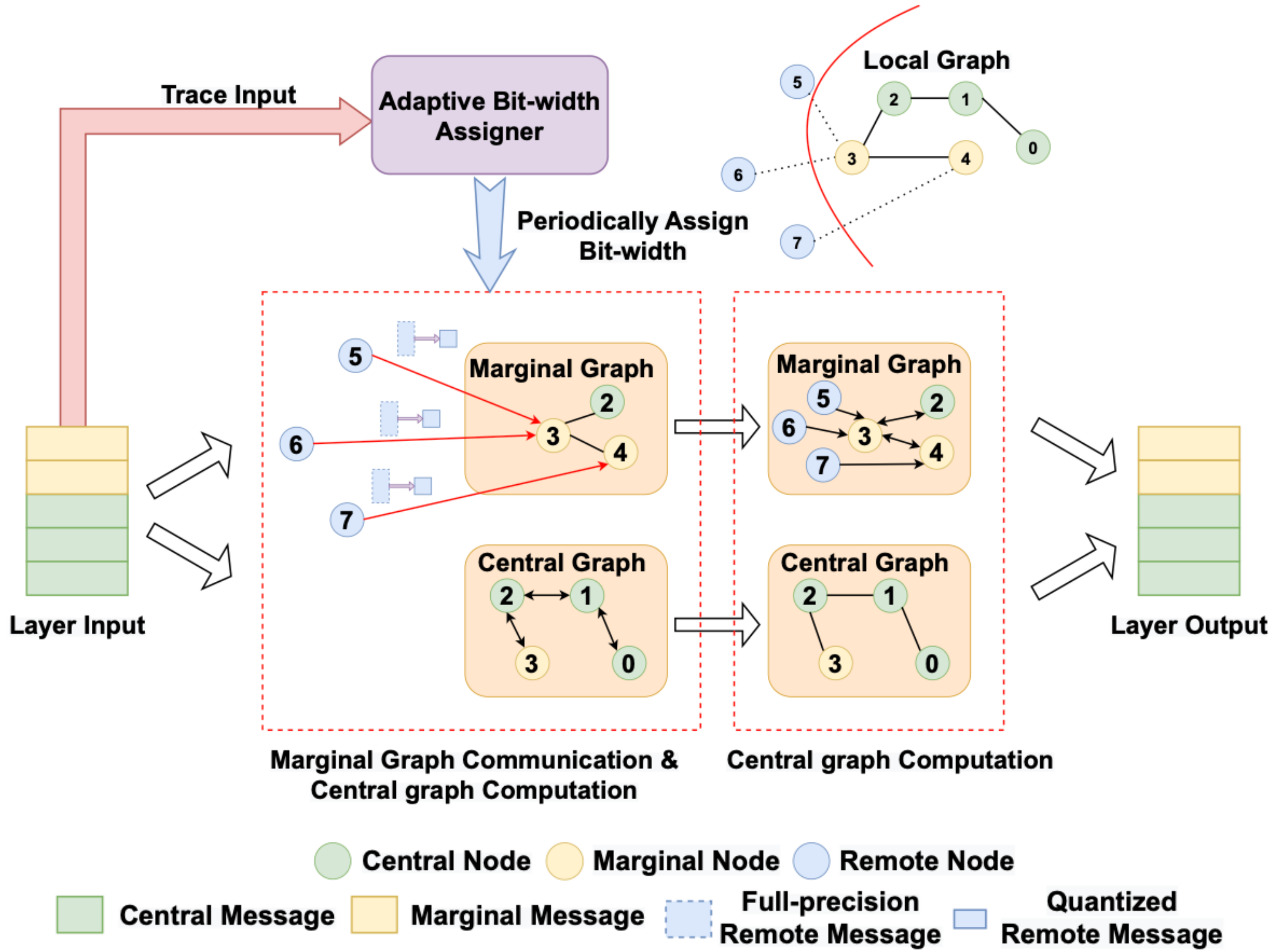
AdaQP



Vanilla Distributed Full-graph Training

- Use **adaptive message quantization** to balance and reduce message exchange sizes among devices
- Hide central nodes computation within marginal nodes communication

Workflow of AdaQP



I. Each local graph is decomposed into central graph and marginal graph; computation of the former overlaps with communication of the latter

II. Each remote message is quantized to certain bit-width set by the **Assigner**

III. The **Assigner** traces input data in each layer and periodically assigns bit-width for remote message quantization

Stochastic Integer Message Quantization

Quantization to compress sending messages

$$\tilde{\mathrm{h}}_{v_b}^{l} = \tilde{q}_b\left(h_v^l\right) = round_{st}\left(\frac{h_v^l - Z_v^l}{S_{v_b}^l}\right)$$

Dequantization to restore received messages

$$\hat{h}_v^l = dq_b\left(\tilde{h}_{v_b}^l\right) = \tilde{h}_{v_b}^l S_{v_b}^l + Z_v^l$$

- h_v^l: message of node v at layer l
- $Z_v^l = \min(h_v^l)$: **zero point**, the minimum value among input vector
- $s_{v_b}^l = \frac{\max(h_v^l) - \min(h_v^l)}{2^{b_v - 1}}$: **scaling factor**, mapping floating point h_v^l to integer $\widetilde{h_{v_b}^l}$

For message vector h_v^l, the reconstructed (after quantization and dequantization) $\hat{h}_v^l$ is:

- **Unbiased estimation** of input: $\mathrm{E}\left[\hat{h}_v^l\right] = h_v^l$
- **Variance bounded** and controlled by bit-width: $Var\left[\hat{h}_v^l\right] = \frac{D_v^l {s_{v_b}^l}^2}{6}$, D_v^l is the dimension of vector h_v^l

Impact of Gradient Variance on Convergence

View full-graph training as empirical risk minimization problem

$$min_{\boldsymbol{w}_t \in R^D} E[\mathcal{L}(\boldsymbol{w}_t)] = \frac{1}{N}\sum_{i}^{N} \mathcal{L}_i(\boldsymbol{w}_t) \quad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \tilde{\boldsymbol{g}}_t$$

Convergence guarantee with gradient variance

Theorem. Suppose our distributed full-graph GNN training runs for T epochs using a fixed step size $\alpha \le \frac{2}{L_2}$*. Select t randomly from {1, · · · , T }. Under Assumption 1, we have*

$$E\left[\left|\left|\nabla\mathcal{L}(\overline{\boldsymbol{w}}_t)\right|\right|^2\right] \le \frac{2(\mathcal{L}(\boldsymbol{w}_1) - \mathcal{L}^*)}{T(2\alpha - \alpha^2 L_2)} + \frac{\alpha L_2 Q^2}{2 - \alpha L_2}$$

- The convergence rate is $O(T^{-1})$, which is **consistent with** vanilla distributed full-graph training
- Training converges to the neighborhood of that of the vanilla distributed full-graph training in solution space, **whose radius is determined by gradient variance upper bound Q**

Connect Quantization to Gradient Variance

Theorem. Given a distributed full-graph (V, E) and optional bit-width set B, for each layer l ∈ *{1,⋯ ,L} in the GNN, the upper bound of the gradient variance Q^l in layer l is:*

$$Q^l = \sum_{v}^{|V|} \left(\sum_{k_1}^{N_R(v)} \sum_{k_2}^{N_R(v)} \alpha_{k1,v}^2 \alpha_{k2,v}^2 \frac{D_{k1}^{l-1} D_{k2}^{l} \left(S_{k1_b}^{l-1} S_{k2_b}^{l}\right)^2}{6} + M^2 \sum_{k}^{N_R(v)} \alpha_{k,v}^2 \frac{D_k^l (S_{k_b}^l)^2}{6} + N^2 \sum_{k}^{N_R(v)} \alpha_{k,v}^2 \frac{D_k^{l-1} (S_{k_b}^{l-1})^2}{6} \right)$$

M and N are constants

recall that: $s_{v_b}^l = \frac{\max(h_v^l) - \min(h_v^l)}{2^{b_v} - 1}$

↑ **Bit-width is here!**

① Size of remote neighbor set $N_R(v)$: decided by graph topology and partition algorithm

② Aggregation coefficient $\alpha_{k,v}$: decided by GNN types (GCN or GraphSAGE)

③ Dimension size D_k^l and value range (numerator) in $S_{k_b}^l$: decided by graph datasets and training process

④ Choices of bit-width b_v: **set it to adjust gradient variance upper bound and communication volume**

Adaptive Bit-width Assignment

Assigner solves a variance-time bi-objective problem to assign bit-width for remote messages sent in each layer, to strike a **convergence-throughput trade-off**

$$min_{b_k \in B, Z>0} \ \lambda \sum_{i}^{N} \sum_{k}^{K_i} \frac{\beta_k}{(2^{b_k}-1)^2} + (1-\lambda)Z, \qquad \lambda \in [0,1]$$

$$s.t._{1 \le i \le N} \ \theta_i \sum_{k}^{K_i} D_k^l b_k + \gamma_i \le Z,$$

added gradient variance bound in each communication round when quantizing sending messages

Communication cost

$$\beta_k = \frac{\sum_v^{N_T(k)} \alpha_{k,v}^2 \, D_k^l (\max(h_k^l) - \min(h_k^l))^2}{6}$$

β_k reflects the total influence of transferring message k between device pair K_i on gradient variance

- K_i: total number of messages transferred in device pair i
- D_k^l: dimension of the remote message vector h_k^l
- θ_i and γ_i: the parameters of cost model

convert the problem to an MILP and invoke an off-the-shelf solver to get the solution

Experimental Evaluation

GNN models: GCN, full-batch GraphSAGE (with mean aggregator)

Baselines: Vanilla distributed full-graph training, PipeGCN, SANCUS

Datasets:

Dataset	#Nodes	#Edges	#Features	#Classes	Size
Reddit	232,965	114,615,892	602	41	3.53GB
Yelp (Zeng et al., 2020)	716,847	6,977,410	300	100	2.10GB
ogbn-products	2,449,029	61,859,140	100	47	1.38GB
AmazonProducts	1,569,960	264,339,468	200	107	2.40GB

Hardware Configurations: 2 servers, each has 4 GPUs

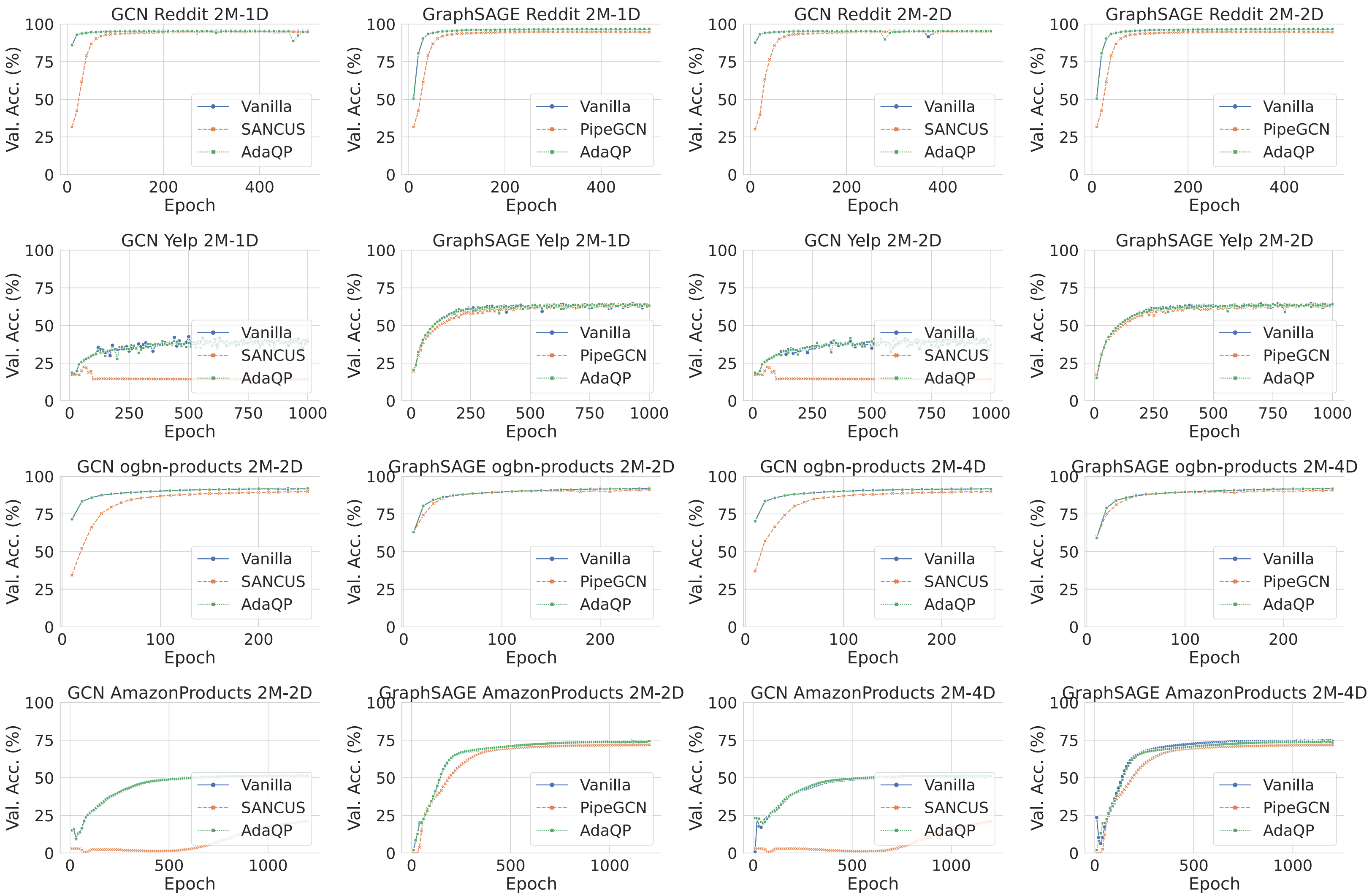
CPU	GPU	Inter-node connection	Intra-node connection
Intel Xeon Gold 6230 2.1GHz	Nvidia Tesla V100 SXM2 32GB	100Gps Ethernet	NVLink

Throughput and Accuracy

Dataset	Partitions	Model	Method	Accuracy(%)	Throughput (epoch/s)	Dataset	Partitions	Model	Method	Accuracy(%)	Throughput (epoch/s)
Reddit	2M-1D	GCN	Vanilla	**95.36±0.03**	0.99	Yelp	2M-2D	GCN	Vanilla	**44.24±0.19**	1.18
			PipeGCN	†	†				PipeGCN	†	†
			SANCUS	94.73±0.17	1.11 (1.12×)				SANCUS	20.75±2.44	0.80
			AdaQP	**95.36±0.02**	**2.17 (2.19×)**				AdaQP	43.96±0.15	**3.04 (2.58×)**
		GraphSAGE	Vanilla	96.50±0.03	0.94			GraphSAGE	Vanilla	64.65±0.08	1.11
			PipeGCN	**96.62±0.00**	**3.72 (3.96×)**				PipeGCN	63.88±0.06	2.63 (2.37×)
			SANCUS	†	†				SANCUS	†	†
			AdaQP	96.49±0.02	2.13 (2.27×)				AdaQP	**64.72±0.13**	**3.15 (2.83×)**
	2M-2D	GCN	Vanilla	95.35±0.04	1.13		2M-4D	GCN	Vanilla	**43.86±0.62**	1.57
			PipeGCN	†	†				PipeGCN	†	†
			SANCUS	94.90±0.02	1.48 (1.31×)				SANCUS	20.78±0.2.45	0.66
			AdaQP	**95.38±0.03**	**2.65 (2.35×)**				AdaQP	43.84±0.63	**3.64 (2.32×)**
		GraphSAGE	Vanilla	96.55±0.03	1.16			GraphSAGE	Vanilla	64.67±0.12	1.19
			PipeGCN	**96.67±0.01**	**3.13 (2.70×)**				PipeGCN	63.73±0.14	2.32 (1.95×)
			SANCUS	†	†				SANCUS	†	†
			AdaQP	96.53 ± 0.04	2.65 (2.28×)				AdaQP	**64.78±0.05**	**3.58 (3.01×)**
ogbn-products	2M-2D	GCN	Vanilla	75.14±0.41	0.61	AmazonProducts	2M-2D	GCN	Vanilla	51.45±0.12	0.42
			PipeGCN	†	†				PipeGCN †	†	
			SANCUS	71.52±0.13	0.26				SANCUS	20.83±0.18	0.32
			AdaQP	**75.32±0.28**	**1.65 (2.70×)**				AdaQP	**51.50±0.08**	**1.16 (2.76×)**
		GraphSAGE	Vanilla	**78.90±0.17**	0.63			GraphSAGE	Vanilla	**75.69±1.32**	0.46
			PipeGCN	77.82±0.01	1.10 (1.75×)				PipeGCN	71.96±0.00	0.99 (2.15×)
			SANCUS	†	†				SANCUS	†	†
			AdaQP	78.85±0.20	**1.67 (2.65×)**				AdaQP	**75.69 ± 1.33**	**1.21 (2.63×)**
	2M-4D	GCN	Vanilla	75.11±0.09	0.79		2M-4D	GCN	Vanilla	51.38±0.16	0.58
			PipeGCN	†	†				PipeGCN	†	†
			SANCUS	71.99±0.16	0.21				SANCUS	21.22±0.07	0.27
			AdaQP	**75.30±0.17**	**2.18 (2.76×)**				AdaQP	**51.56±0.20**	**1.60 (2.76×)**
		GraphSAGE	Vanilla	78.89±0.09	0.77			GraphSAGE	Vanilla	75.80±1.16	0.62
			PipeGCN	76.67±0.01	1.10 (1.43×)				PipeGCN	71.91±0.00	1.02 (1.65×)
			SANCUS	†	†				SANCUS	†	†
			AdaQP	**78.90±0.08**	**2.15 (2.79×)**				AdaQP	**75.98±1.18**	**1.61 (2.60×)**

AdaQP achieves highest throughput in most cases (**2.19~3.01X** faster than Vanilla) & comparable accuracy (within **- 0.30% ~ + 0.19%** of Vanilla’s accuracy)

Convergence Curve



AdaQP preserves $O(T^{-1})$ convergence rate, verifying theoretical results

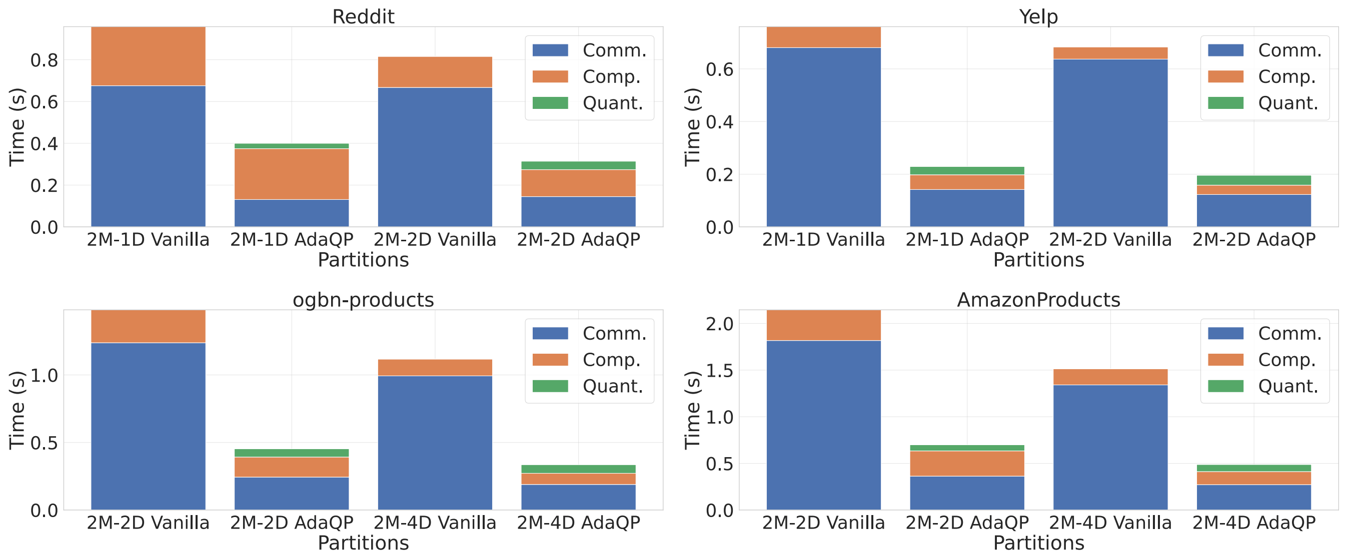
Accuracy-Throughput Trade-off and Time Breakdown

Adaptive quantization vs. uniform random bit-width sampling

Partitions	Model	Method	Accuracy (%)	Throughput (epoch/s)
2M-2D	GCN	Uniform	75.03±0.36	1.70
		Adaptive	**75.32±0.28**	1.65
	GraphSAGE	Uniform	78.84±0.23	1.64
		Adaptive	**78.85±0.20**	1.67
2M-4D	GCN	Uniform	75.16±0.16	2.14
		Adaptive	**75.30±0.17**	2.18
	GraphSAGE	Uniform	78.85±0.08	2.07
		Adaptive	**78.90±0.08**	2.15

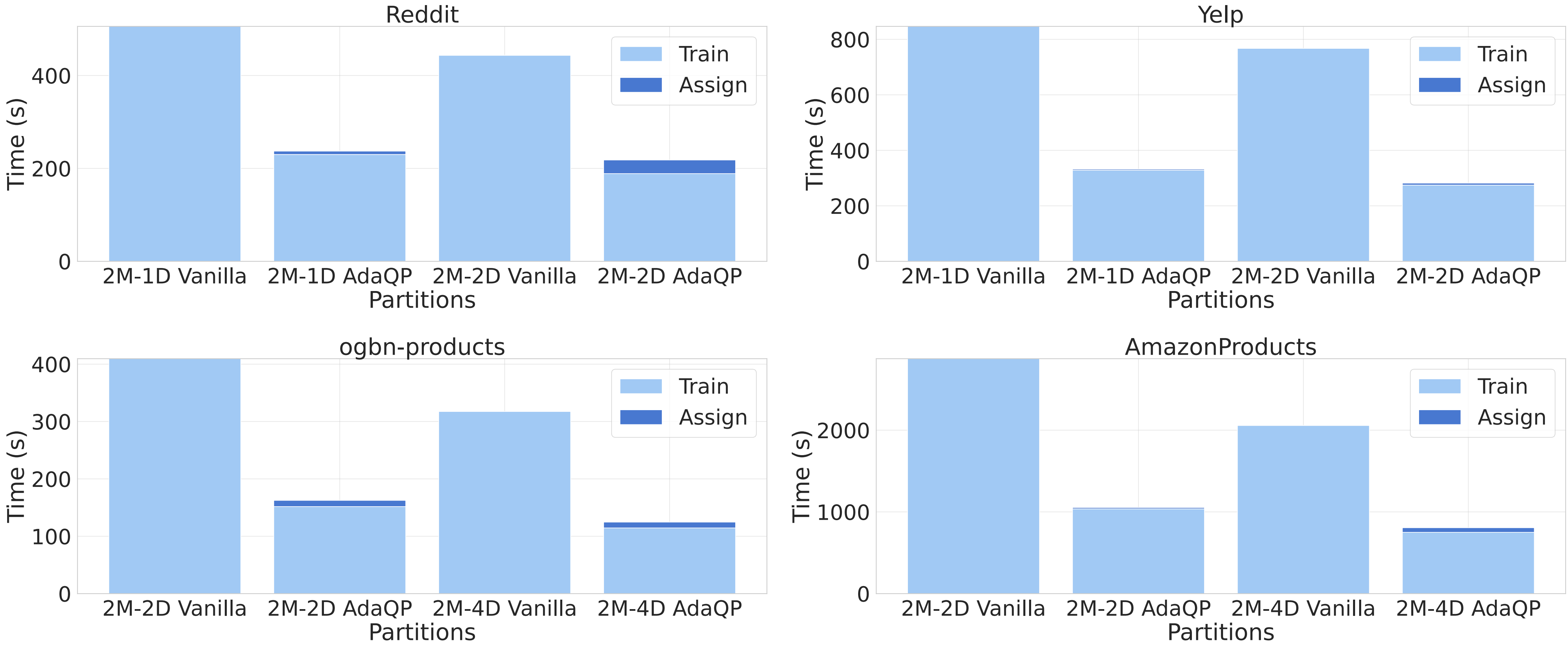
AdaQP consistently achieves higher accuracy than random sampling with close throughput, striking **better accuracy-throughput trade-off**

Per-epoch training time breakdown



quantization/dequantization overhead of AdaQP is small (**5.53%~13.88%**)

End-to-end Wall-clock Time Breakdown



Compared to wall-clock time reduction, bit-width assignment overhead is negligible (**5.43%** in average)

Summary

- Two key designs:
  - Message quantization with time-variance-aware bit-width assignment
  - Communication-computation parallelization on each local graph
- Convergence guarantee and theoretical analysis on relationship between message quantization and gradient variance
- Reduce communication cost by 79.98% on average and achieve 2.19~3.01X training throughput improvement

Takeaway: GNNs are robust to quantization, even when loss compression is performed on all messages (features, embeddings, embedding gradients) of each GNN layer

Thank You

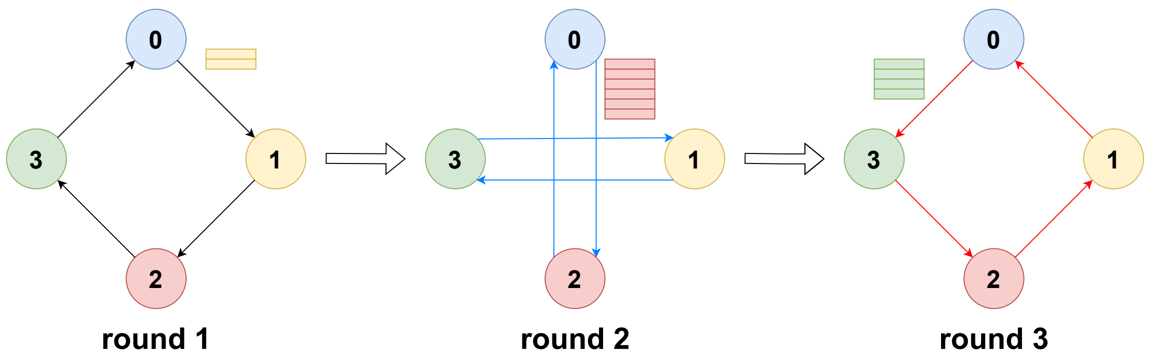
The Artifact of AdaQP:

https://doi.org/10.5281/zenodo.7783787

Contact:

wanborui@connect.hku.hk

Message Transferring Modeling



Implement message exchange with **Ring all2all**

Objective 1: minimize struggler in each communication round

$$min_{b_k \in B} max_{1 \le i \le N} \theta_i \sum_{i}^{K_i} D_k^l b_k + y_i$$

- K_i: total number of messages transferred in device pair i
- θ_i and γ_i: the parameters of cost model
- B: the optional bit-width set, set to [2,4,8] in our work

Variance Upper Bound Modeling

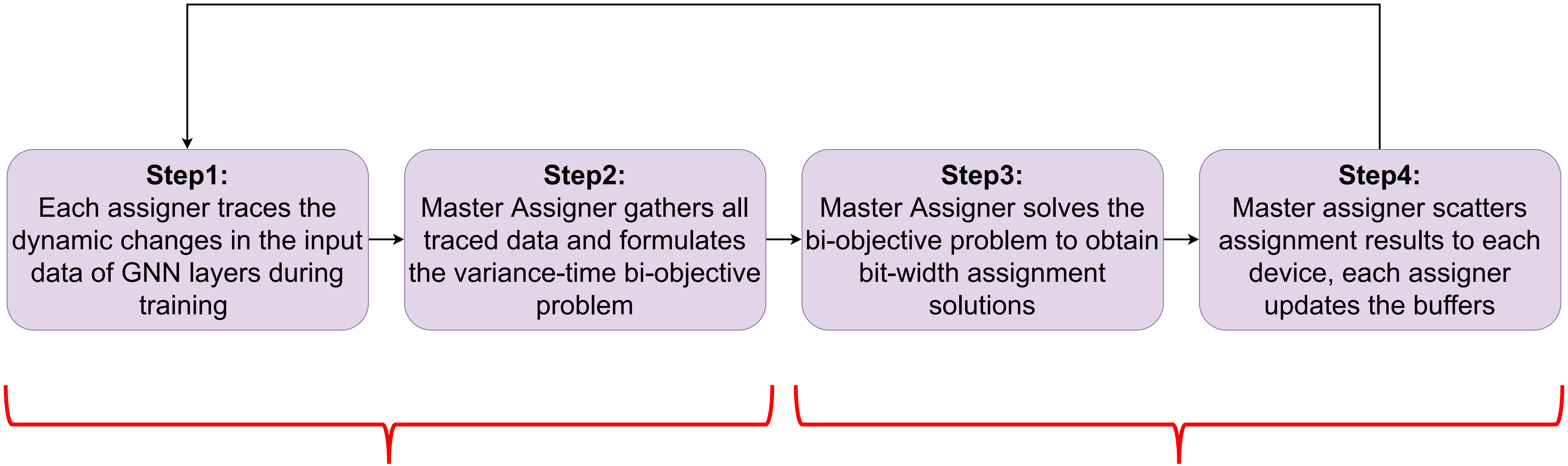
Objective2: minimize **added variance upper bound** in each communication round due to message quantization

$$min_{b_k \in B} \sum_{i}^{N} \sum_{k}^{K_i} \frac{\beta_k}{(2^{b_k} - 1)^2}$$

$$\beta_k = \frac{\sum_v^{N_T(k)} \alpha_{k,v}^2 \, D_k^l (\max(h_k^l) - \min(h_k^l))^2}{6}$$

$N_T(k)$ denotes k' s neighbors in the target device to which k' s message is to be sent

Adaptive Bit-width Assignment



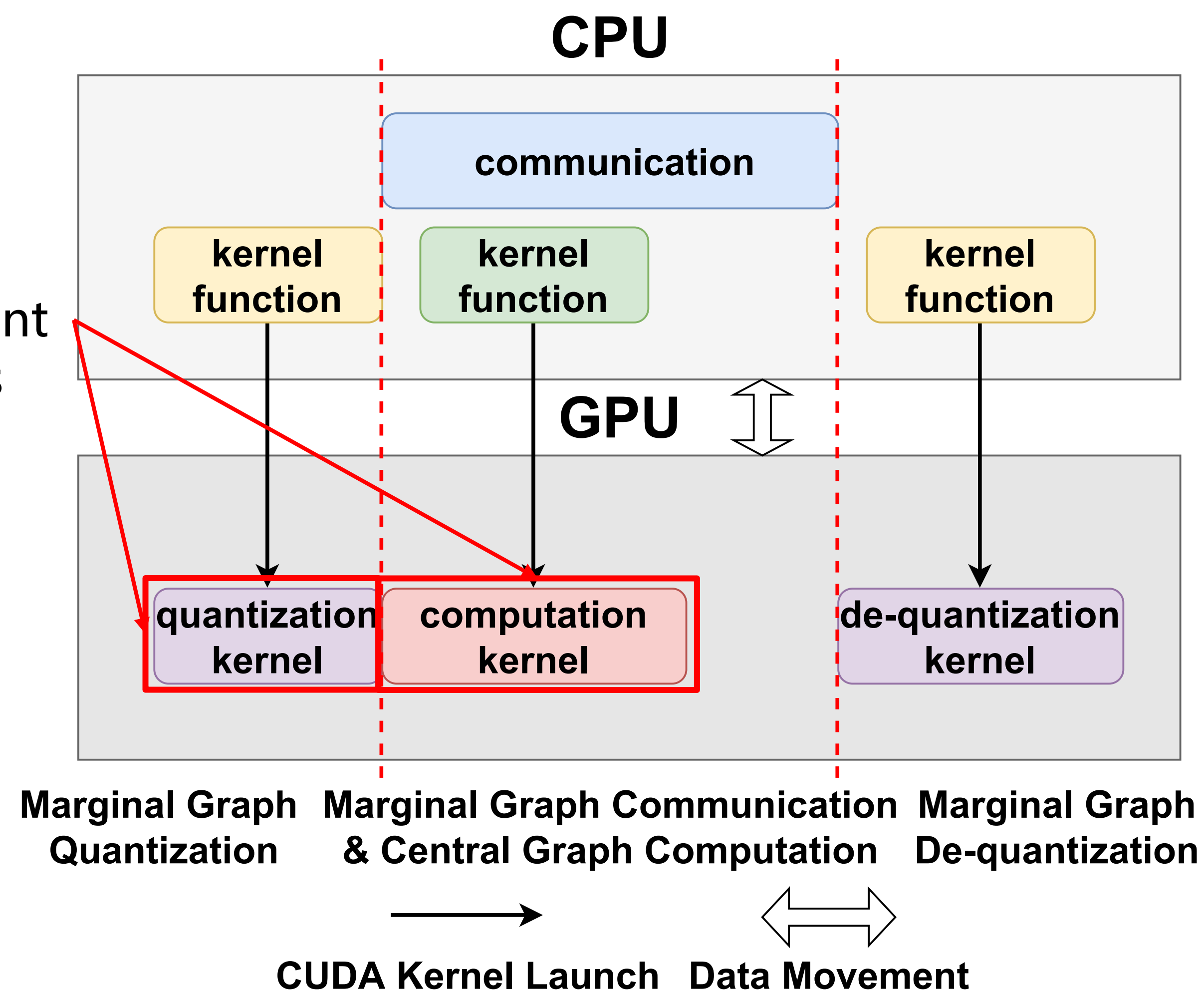
Each worker gather needed data and periodically send it to Master, then waiting for results

Master assigns bit-width for remote messages sent in each layer and dispatches the results

Solve the variance-time bi-objective problem to strike a convergence-efficiency trade-off

GPU Resources Isolation

Quantization Kernel and Computation Kernel content for GPU compute resources



Control CUDA kernel launching time to avoid GPU resources contention

End-to-end Wall-clock Time Comparison

Dataset	Partitions	Model	Method	Wall-clock Time (s)	Dataset	Partitions	Model	Method	Wall-clock Time (s)
Reddit	2M-1D	GCN	Vanilla	505.79	Yelp	2M-2D	GCN	Vanilla	846.79
			PipeGCN	†				PipeGCN	†
			SANCUS	447.28				SANCUS	1249.89
			AdaQP	**237.24**				AdaQP	**332.37**
		GraphSAGE	Vanilla	530.46			GraphSAGE	Vanilla	897.28
			PipeGCN	**135.29**				PipeGCN	381.11
			SANCUS	†				SANCUS	†
			AdaQP	246.71				AdaQP	**321.54**
	2M-2D	GCN	Vanilla	443.53		2M-4D	GCN	Vanilla	767.47
			PipeGCN	†				PipeGCN	†
			SANCUS	335.56				SANCUS	1509.41
			AdaQP	**218.14**				AdaQP	**281.77**
		GraphSAGE	Vanilla	429.85			GraphSAGE	Vanilla	839.96
			PipeGCN	**159.66**				PipeGCN	430.63
			SANCUS	†				SANCUS	†
			AdaQP	208.34				AdaQP	**289.23**
ogbn-products	2M-2D	GCN	Vanilla	409.54	AmazonProducts	2M-2D	GCN	Vanilla	2874.77
			PipeGCN	†				PipeGCN	†
			SANCUS	940.16				SANCUS	3782.44
			AdaQP	**162.53**				AdaQP	**1053.51**
		GraphSAGE	Vanilla	397.91			GraphSAGE	Vanilla	2597.21
			PipeGCN	229.11				PipeGCN	1212.65
			SANCUS	†				SANCUS	†
			AdaQP	**155.94**				AdaQP	**1008.34**
	2M-4D	GCN	Vanilla	317.48		2M-4D	GCN	Vanilla	2057.70
			PipeGCN	†				PipeGCN	†
			SANCUS	1186.68				SANCUS	3880.68
			AdaQP	**124.67**				AdaQP	**806.29**
		GraphSAGE	Vanilla	326.05			GraphSAGE	Vanilla	1927.85
			PipeGCN	229.31				PipeGCN	1171.38
			SANCUS	†				SANCUS	†
			AdaQP	**133.93**				AdaQP	**771.52**

AdaQP Achieve highest shortest wall-clock time (14/16) in most cases

Takeaway

- Empirically, GNN training is robust to stochastic integer quantization, even when it is performed to each part and each layer of GNN
- Stochastic integer quantization can reduce the data transferring overhead in all kinds of GNN training systems