G-HEMP: Fast Multi-GPU Private Inference for Large-Scale GCNs

Ran Ran, Zhaoting Gong, Zhaowei Li, Xianting Lu, Jiajia Li, Wujie Wen
North Carolina State University

Intro: Private GCN inference

Why GCN privacy matters

GCNs are used on graph data from social networks, recommendations, finance, healthcare, and scientific discovery.

- Both node features X and graph topology A can reveal sensitive information.
- Cloud inference is useful, but plaintext graph upload is often unacceptable.
- Homomorphic Encryption (HE) lets the server compute directly on encrypted data.

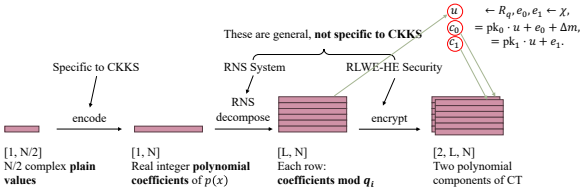
Goal: make encrypted GCN inference practical on GPU systems.

Threat model

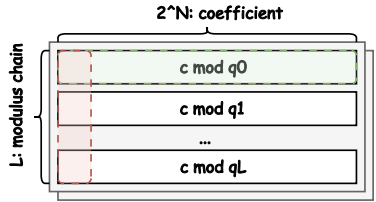
- Client encrypts A and X.
- Cloud hosts plaintext model weights W.
- Server returns encrypted output.
- Client decrypts locally.

$$\mathrm{GCN}(A, X, W)$$
$$\Downarrow$$
$$\mathrm{GCN}(\mathrm{Enc}(A), \mathrm{Enc}(X), W)$$

HE background: From Data to Ciphertext Representation



(a) Data encoding and encryption into CKKS ciphertext



(b) RNS representation of each polynomial

Polynomial for SIMD

A ciphertext is not a simple vector. CKKS first encodes packed values into a polynomial, then represents each polynomial under multiple RNS moduli. This makes computation SIMD-friendly, but also introduces memory expansion and modulus-level data movement.

Plaintext vs. HE Matrix Multiplication

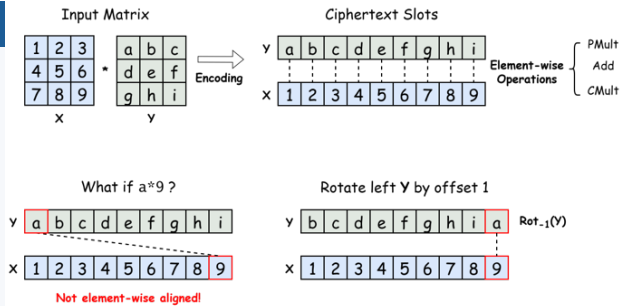
Plaintext matrix multiplication

$$\mathbf{A} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 2 & 9 \\ 4 & 5 & 6 \\ 7 & 8 & 3 \end{bmatrix}$$

Different output entries require different pairings:

$$C_{1,1} = a \cdot 1 + b \cdot 4 + c \cdot 7$$

$$C_{1,3} = a \cdot 9 + b \cdot 6 + c \cdot 3$$



In plaintext, these pairings are directly indexed.

In HE, packed slots must be aligned through rotations.

The arithmetic is simple, but HE introduces extra slot movement to realize different matrix-multiplication alignments.

Where the cost comes from

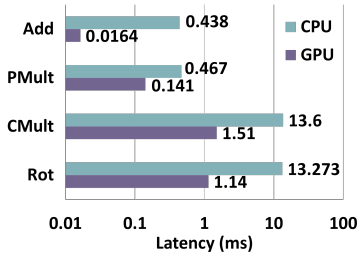
GCN layer

$$H = \sigma(AXW)$$

- XW: feature aggregation.
- AX: node aggregation over graph structure.
- In this work, both A and X are encrypted.

CKKS Operations

Ciphertexts pack many slots, so layout decides whether HE operations are efficient or wasteful.



Challenge: GPU primitives are not enough

C1. Packing-memory conflict

Challenge: prior packing reduces rotations by duplicating encrypted A.

- A scales with node count.
- Existing packing may replicate A across feature slots.
- This is fatal on GPUs with limited memory.

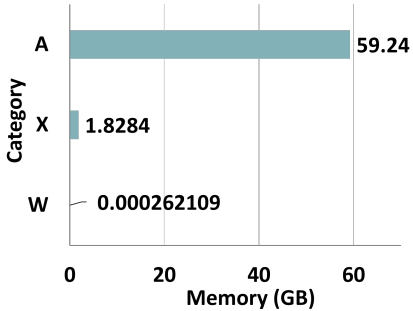
C2. Multi-GPU dependency conflict

Challenge: naive partitioning cuts inside ciphertexts.

- Key-switching needs complete ciphertext state.
- Limb-level splitting creates inter-GPU dependency.
- More GPUs can increase latency.

Problem	What breaks	Needed design
Encrypted A is huge	GPU memory capacity	duplication-free packing
KSO is ciphertext-coupled	multi-GPU communication	ciphertext-level graph partition

Challenge 1: encrypted adjacency dominates memory



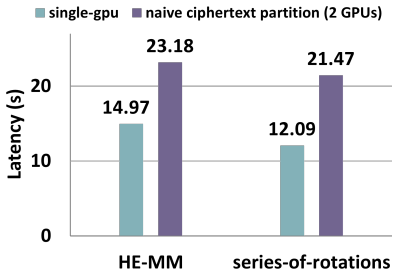
Observation

On PubMed, a single encrypted adjacency matrix is about **59 GB**; it is much larger than encrypted features and plaintext weights.

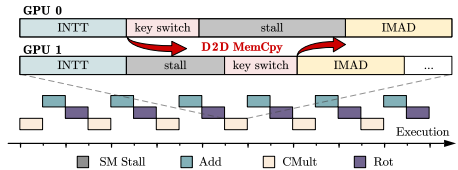
- CPU DRAM can hide this problem for some settings.
- A single GPU may not fit A, X, and intermediate rotated copies.
- Duplicating A by the number of packed features quickly becomes impossible.

Method: pack data so A is never feature-duplicated.

Challenge 2: naive multi-GPU can be slower



Limb-level ciphertext partitioning adds communication during HE operations.

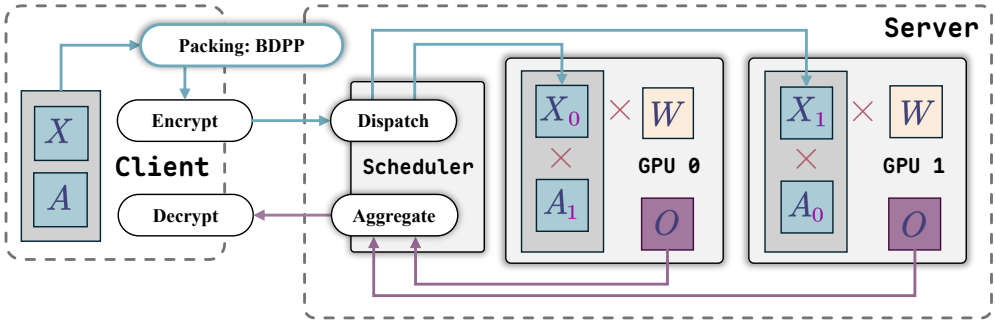


Key point

Key-switching in *CMult* and *Rotation* is not an element-wise operation. Splitting a ciphertext across GPUs puts communication on the critical path.

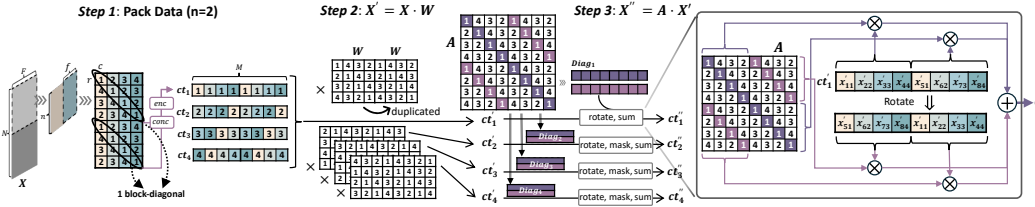
Method: partition at complete-ciphertext granularity, aligned with graph blocks.

Method overview: G-HEMP



1. Block-Diagonal Parallel Packing	2. Graph Partition with X-sharing
Reduce rotations while eliminating encrypted adjacency duplication.	Split A and X across GPUs at ciphertext granularity, then aggregate once.

Method 1: Block-Diagonal Parallel Packing



Idea

For each $f \times f$ block, extract same-index diagonals from multiple node blocks and concatenate them into one ciphertext.

- Each packed block mixes different nodes and features.
- XW can use SIMD slots efficiently.
- AX uses matching block-diagonal A ciphertexts.
- No $f\times$ duplication of encrypted A.

$$Rot_{\text{total}} = ct_{\text{total}}\big((n-1)+2(f-1)\big)$$

Method 1: Block-Diagonal Parallel Packing

Before: rotation-memory tradeoff

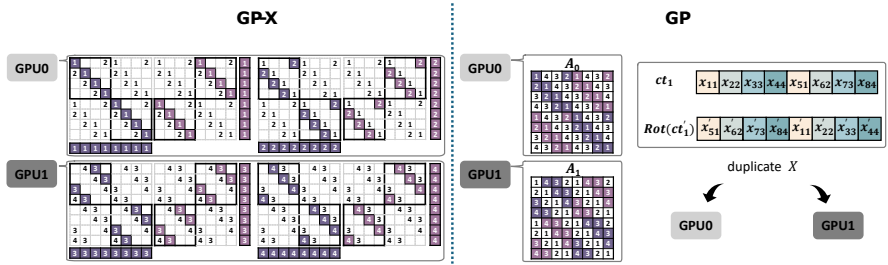
- Feature-wise packing: low slot utilization, many rotations.
- Penguin: fewer rotations, but duplicates encrypted A.
- Large graphs hit GPU memory limit.

After: *BDPP*

- Matches Penguin's asymptotic rotation complexity.
- Reduces memory cost for A from $O(nf)$ to $O(n)$.
- Keeps enough parallelism for GPU execution.

Packing method	Rotation complexity	Memory cost
Gazelle-style	$O(n \cdot 2f)$	$O(n \cdot 2f)$
Penguin	$O(n + 2f)$	$O(nf)$
BDPP	$\mathbf{O(n + 2f)}$	$\mathbf{O(n)}$

Method 2: Graph Partition with X-sharing



Partitioning rule

The minimum partition unit is one complete ciphertext.

- Split encrypted A to avoid adjacency duplication.
- Split encrypted X to avoid duplicated rotations.
- Duplicate only W, because it is plaintext and tiny.
- Exchange compact partial results after local HE computation.

$$(A_0 + A_1)(X_0 + X_1)W$$

Results: single-GPU packing efficiency

Table: single-GPU packing comparison. Speedup is normalized to feature-wise packing.

Dataset	Method	Memory (GB)	Latency (s)	Speedup
Amazon Photo	FWP	19.31	854.20	1.00
	Penguin + dup	14.81	3253.53	0.263
	BDPP	**14.15**	**193.79**	**4.41**
Amazon Computers	FWP + re-rot	38.76	3210.23	1.00
	Penguin + dup	38.72	10148.27	0.316
	BDPP	**37.83**	**750.14**	**4.28**
PubMed	FWP + re-rot	OOM	–	–
	Penguin + dup	OOM	–	–
	BDPP	OOM	–	–

- *BDPP* avoids the two bad single-GPU choices: many re-rotations in FWP or duplicated encrypted adjacency in Penguin.
- On Amazon Photo/Computers, *BDPP* gives **4.41**×/**4.28**× speedup with similar or lower memory.
- PubMed already exceeds one-GPU capacity, motivating graph-aware multi-GPU partitioning.

Results: multi-GPU policy matters

Table: two-GPU policy comparison. Speedup is normalized to each method's single-GPU latency.

Dataset	Packing	Method	Comm. (GB)	Mem./GPU (GB)	Latency speedup
Amazon Photo	Penguin	LLP	41.7	7.41	0.322
		GP	0.03	11.41	1.800
		GP-X	0.03	9.65	1.919
	BDPP	LLP	28.8	7.07	0.642
		GP	0.04	11.41	1.86
		GP-X	**0.04**	**7.07**	**1.902**
Amazon Computers	Penguin	LLP	83.3	18.91	0.322
		GP	0.06	24.07	1.860
		GP-X	0.06	21.83	1.910
	BDPP	LLP	57.6	18.91	0.64
		GP	0.07	24.07	1.912
		GP-X	**0.07**	**18.91**	**1.934**
PubMed	Penguin	all policies	–	OOM	–
	BDPP	LLP	86.4	37.60	0.639
		GP	–	OOM	–
		GP-X	**0.11**	**37.60**	**1.98**

- **GP-X + BDPP** is the only setting that stays close to the ideal 2-GPU target: about **50%** per-GPU memory and up to **1.98**$\times$ speedup.
- LLP cuts ciphertext limbs, so key-switching communication lands on the critical path and can be slower than one GPU.
- GP avoids large communication, but duplicating X also duplicates rotations and ciphertext copies; GP-X partitions both A and X.

Results: scaling to four A100 GPUs

Table: scaling from 2 to 4 A100 GPUs under BDPP. Speedup is relative to one GPU.

Dataset	Method	GPUs	Latency (s)	Speedup
Amazon Photo	LLP	2	163.54	0.70
	GP-X	2	**60.12**	**1.93**
	LLP	4	387.38	0.30
	GP-X	4	**32.03**	**3.63**
Amazon Computers	LLP	2	633.31	0.70
	GP-X	2	**229.80**	**1.96**
	LLP	4	1500.16	0.30
	GP-X	4	**118.80**	**3.79**
PubMed	LLP	2	1391.46	0.70
	GP-X	2	**498.63**	**1.98**
	LLP	4	3296.04	0.30
	GP-X	4	**254.97**	**3.88**

- *GP-X* continues to scale on 4 GPUs, reaching **3.63**×–**3.88**× speedup across all benchmarks.
- LLP degrades to **0.30**× because limb-dependent key switching creates cross-GPU synchronization bottlenecks.
- Since *GP-X* partitions independent ciphertext groups, communication is decoupled from HE internals and scaling is mainly limited by input granularity.

Conclusion

What *G-HEMP* contributes

- First systematic multi-GPU framework for HE-GCN inference with encrypted A and X.
- *BDPP* removes encrypted adjacency duplication.
- *GP-X* partitions graph-aligned ciphertext groups across GPUs.

Main results

- Up to **4.41**× single-GPU speedup over feature-wise packing.
- Up to **1.98**× speedup on 2 GPUs.
- Up to **3.88**× speedup on 4 GPUs.
- Outperforms limb-level multi-GPU HE partitioning by up to **3.13**×.

Layout-aware packing + graph-aware partitioning makes private GCN inference scalable.

Thank you

Questions?

G-HEMP: Fast Multi-GPU Private Inference for Large-Scale GCNs