DOLPHIN

Scalable Disk–RAM–GPU Pipelined Training for Massive Temporal GNNs

Zezhong Ding Rui Guo Wenbo Zhen
Junlin Lv Jianliang Xu Xike Xie

University of Science and Technology of China
Hong Kong Baptist University

SC26 Chicago November 2026



Data Darkness Laboratory

Background

Temporal (Dynamic) Graph Learning is Widely Applied

Business Intelligence [1], Molecular Dynamics [2], and Cell Biology [3]

Prediction in World Models [4]

Cause: Dynamic graphs are massive, complex and ever-expanding.

Consequence: Scaling these applications exposes two training challenges.

Challenge 1 Training Efficiency

Frequent data transfers stall GPUs
Temporal dependencies limit parallelism

Challenge 2 Memory Scalability

Growing edge histories inflate RAM use
Preloading all features limits graph scale

***Question:* How can we scale dynamic graph learning with high runtime efficiency and low memory consumption?**

Limitations in Existing T-GNN Training Systems

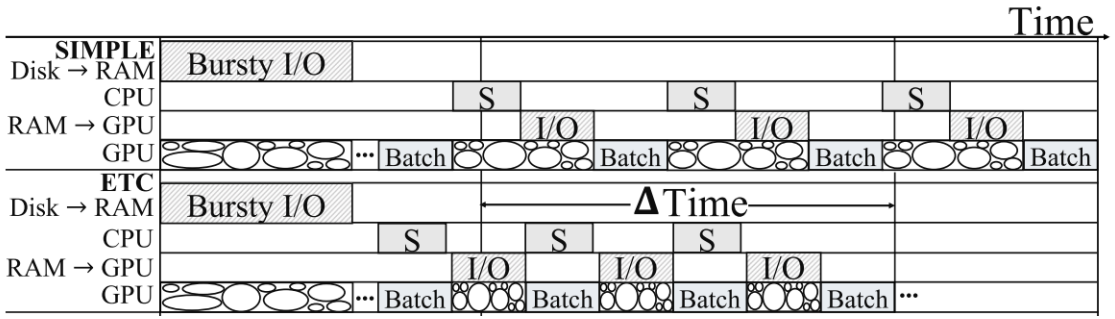
Current T-GNN systems typically employ a RAM-GPU training pipeline, assuming graph data can reside in CPU-memory.

RAM-GPU training pipeline (**three stages**)

(1) Sampling: CPU cores collect batches and their neighbors

(2) I/O (RAM-to-GPU): Batches are transferred from RAM to GPU.

(3) Training

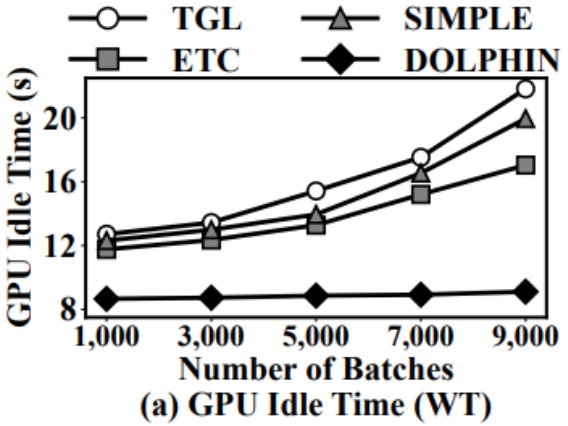


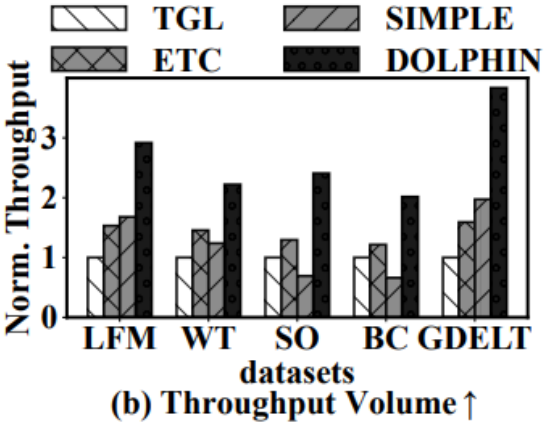
Weakness (Low Runtime Efficiency)

This pipeline suffers from frequent RAM–to-GPU transfers.

(a) GPU idle time scales with the number of batches. (Figure (a))

(b) High GPU idle time leads to low throughput. (Figure (b))





Limitations in Existing T-GNN Training Systems

Weakness (Low RAM Efficiency)

Preloading all features makes RAM grow with graph history

System	Existing optimization	Remaining limitation
TGL	CPU sampling and batch-based training	Repeated PCIe transfers expose GPU stalls at batch boundaries
ETC	Sampling–training overlap and local batch expansion	Batch-boundary I/O persists Local batching misses wider dependencies
SIMPLE	Feature caching to reduce RAM–GPU traffic	Cache recomputation under distribution shifts Transfer gaps remain between batches
PipeTGL	Pipelined execution across GPUs	Distributed optimizations leave intra-node RAM–GPU data movement unresolved

RAM–GPU pipelines improve local stages, but retain system-level bottlenecks

Therefore, we need to design a new pipeline to address these challenges.

DOLPHIN Framework: Disk–RAM–GPU Pipelined Training

Temporal GNN Training runs for multiple epochs, each divided into batches

For each batch: sample node neighbors, then update the model

Before training (lightweight preprocessing)

Offline sampling

Sample neighbors and group batches into chunks

Feature deduplication

Remove duplicate features within each chunk

Incremental storage

Store only changes between consecutive chunks on disk

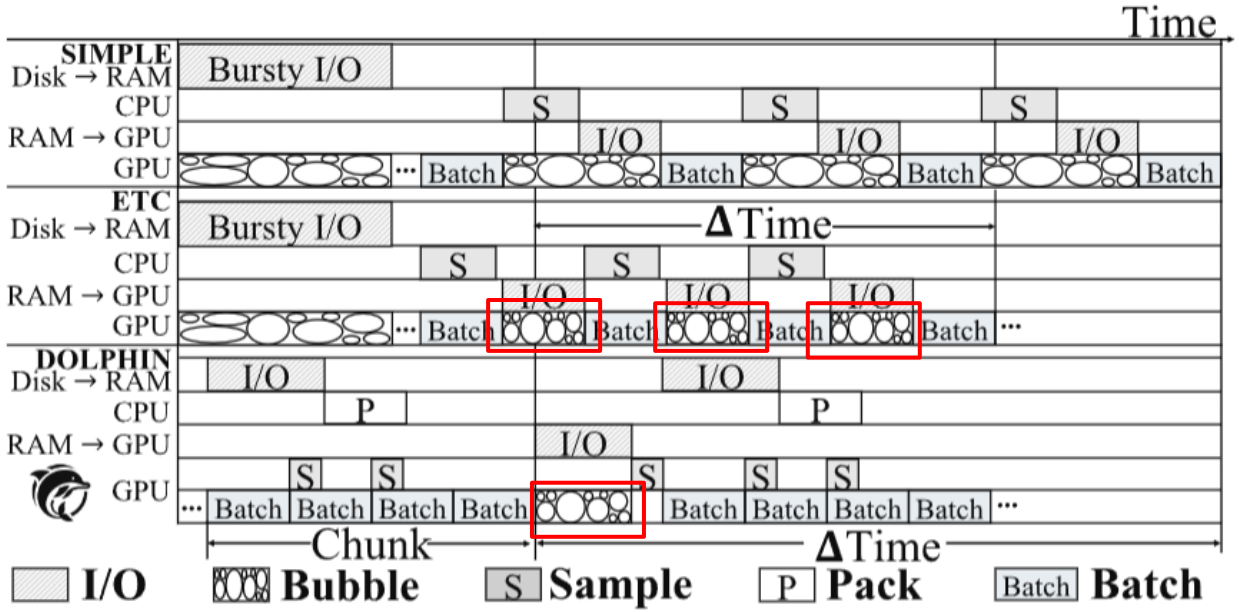
During training (four stages)

(1) I/O (Disk-to-RAM): Incrementally load only the required chunks into RAM.

(2) Transforming: Transform chunks into training batches.

(3) I/O (RAM-to-GPU): Transfer the training batches from RAM to GPU.

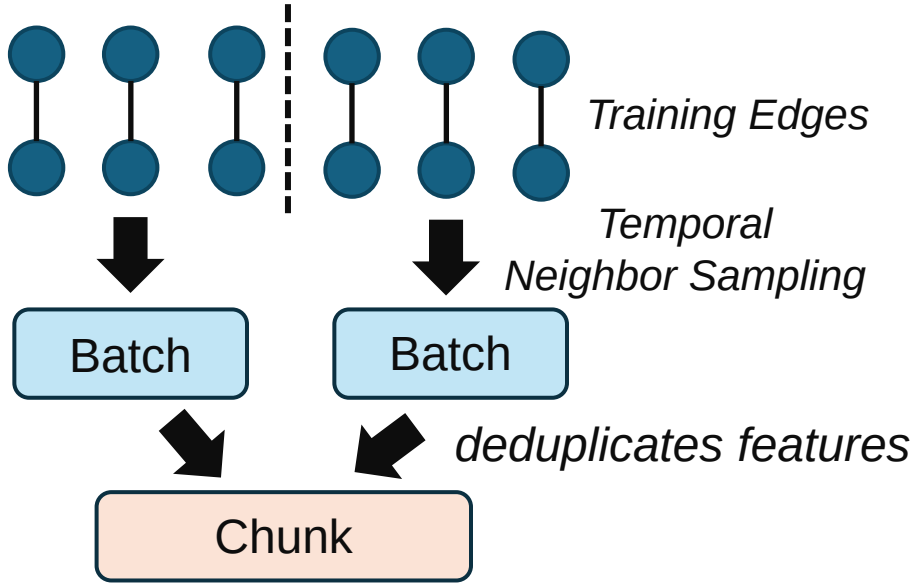
(4) GPU training

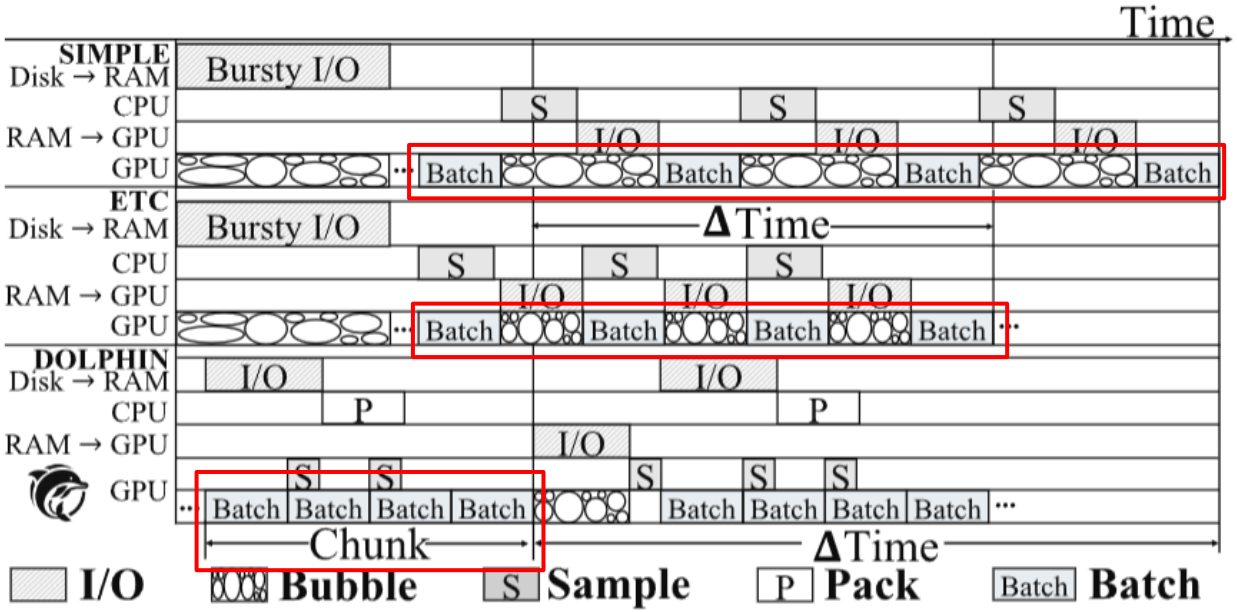


Overlap GPU training with chunk loading, batching, and transfers

DOLPHIN: A Chunk-Based Disk–RAM–GPU Pipelined System

Chunk-Base Optimization: **aggregate multiple batches and their neighbors into chunks**





(1) DOLPHIN deduplicates features across batches within a chunk → ***Reduce task execution time***

(2) DOLPHIN replaces frequent batch-triggered transfers with fewer, chunk-triggered ones → ***Eliminating idle stalls***

DOLPHIN: A Chunk-Based Disk–RAM–GPU Pipelined System

Chunk-Base Optimization: **aggregate multiple batches and their neighbors into chunks**

(1) DOLPHIN *deduplicates features across batches within a chunk* → ***Reduce task execution time***

(2) DOLPHIN replaces frequent batch-triggered transfers with fewer, chunk-triggered ones → ***Eliminating idle stalls***

This optimization defines the chunk as the fundamental scheduling unit, but chunking alone is insufficient.

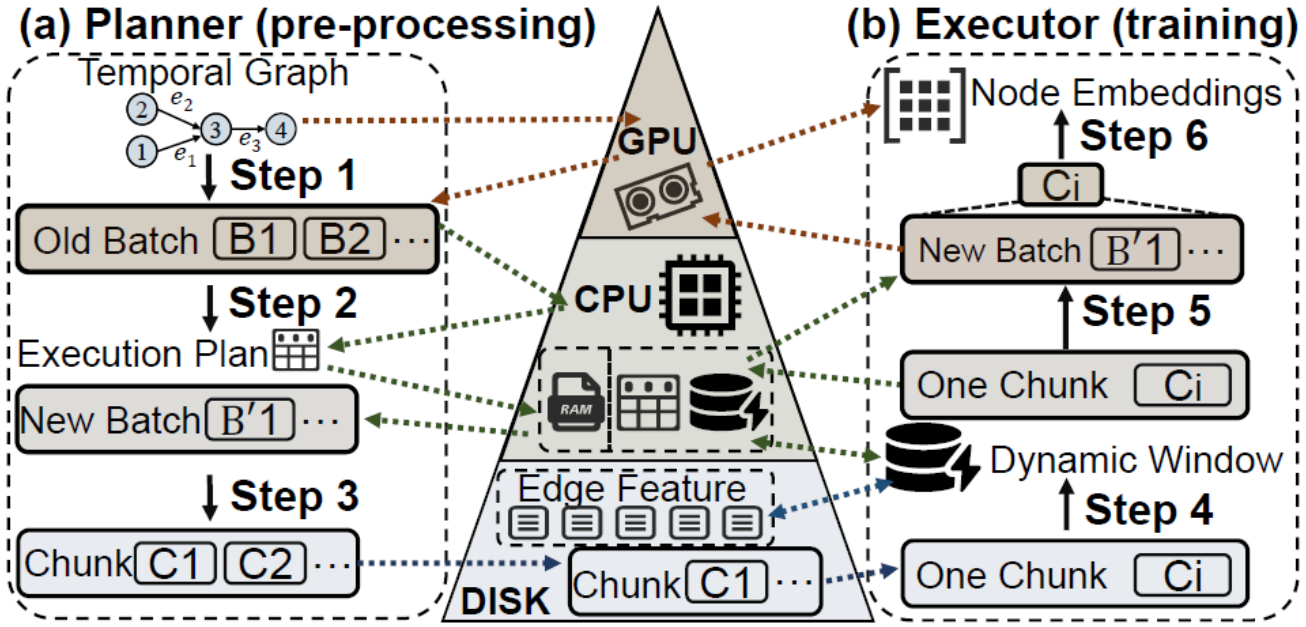
I/O Bottleneck: we should provide I/O (Disk-to-RAM) at the right granularity.

Over-provisioning wastes RAM;
Under-provisioning leaves GPU kernels idle

Compute Bottleneck: We should construct training batches that respect temporal dependency constraints.

Oversized batches degrade model quality;
Undersized batches leave GPUs underutilized

DOLPHIN Framework



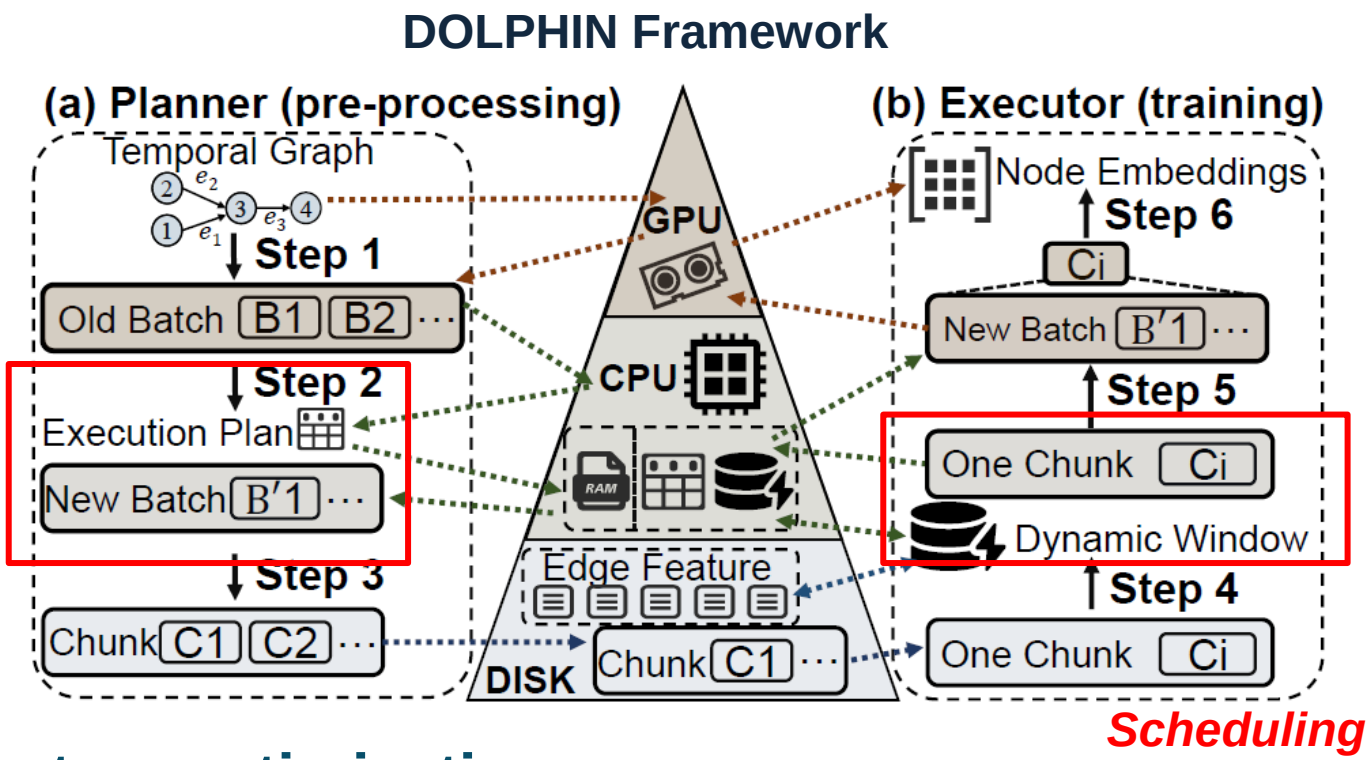
DOLPHIN: A Chunk-Based Disk–RAM–GPU Pipelined System

I/O Bottleneck: We should provide I/O (Disk-to-RAM) at the right granularity.

Over-provisioning wastes RAM;
Under-provisioning leaves GPU kernels idle

Compute Bottleneck: We should construct training batches that respect temporal dependency constraints.

Oversized batches degrade model quality;
Undersized batches leave GPUs underutilized



Making this pipeline efficient requires two optimizations.

(1) To address I/O Bottleneck, we propose I/O-Efficient Feature Window Scheduling.

Keep the features needed for current training to reduce RAM usage and keep GPUs supplied with data.

(2) To address Compute Bottleneck, we propose Dependency Graph-based Execution Planning.

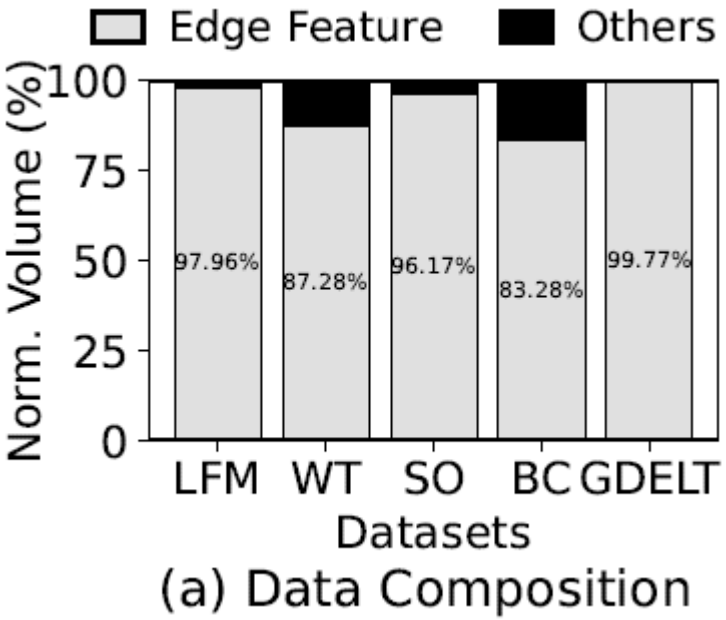
Parallelize independent training edges to improve GPU utilization while preserving temporal dependencies.

Optimization 1: I/O-Efficient Feature Window Scheduling

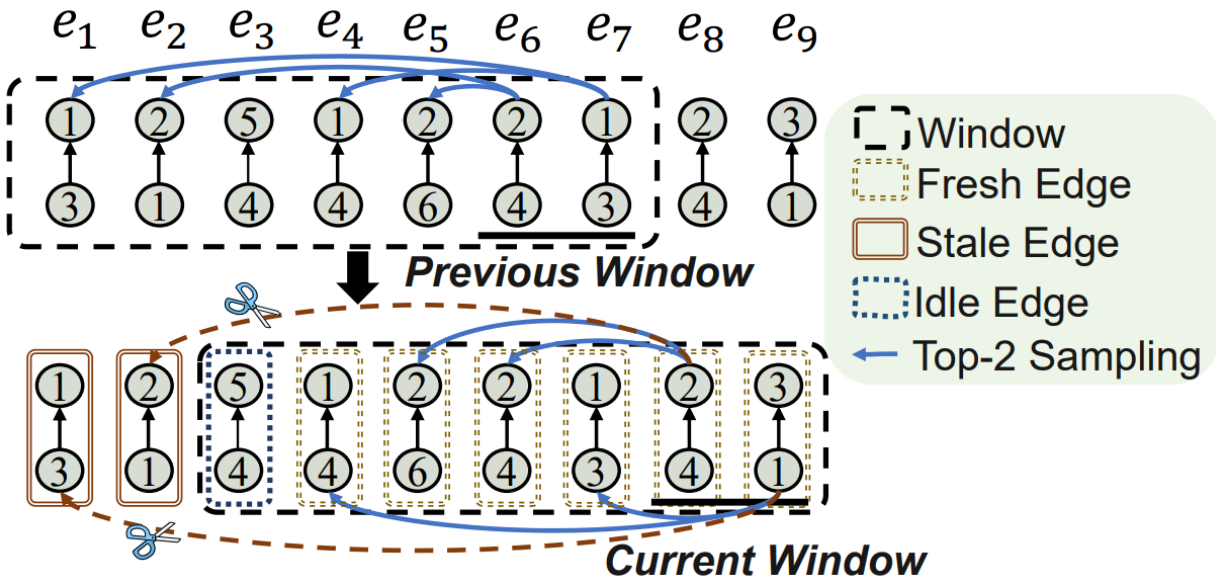
Motivation

Edge features are also the primary source of I/O (Disk-to-RAM) (Figure (a))

We need to optimize feature scheduling first.



(a) Data Composition

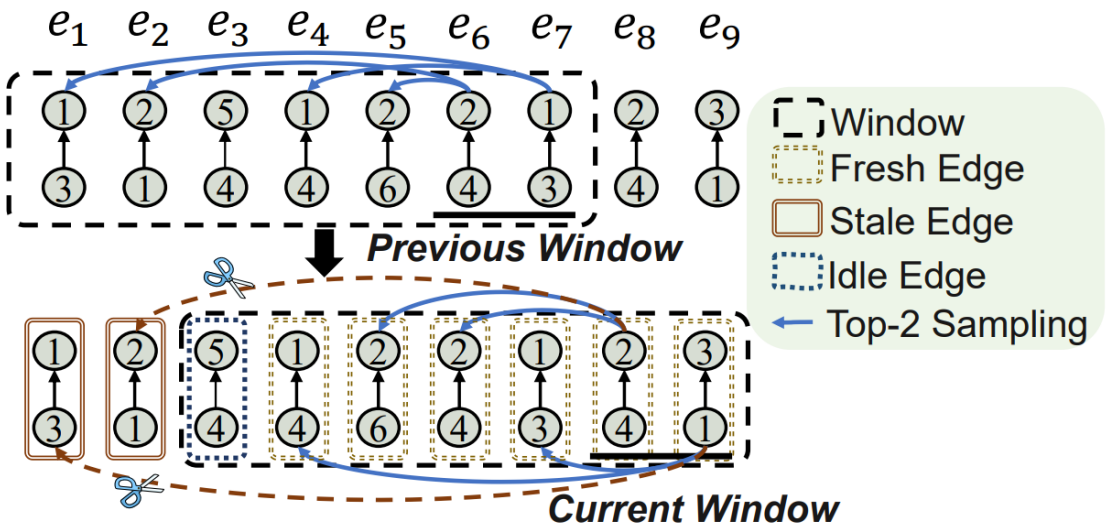


Fresh Edge Features: the edge features directly required in the current chunk

Stale Edge Features: edge features that are not among its top-α most recent neighbors that will **not** be used again during the current training epoch

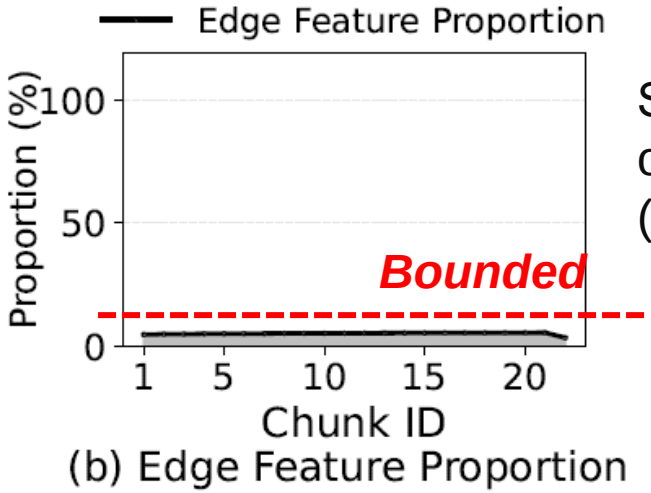
Optimization 1: I/O-Efficient Feature Window Scheduling

Dynamic RAM Window Mechanism



As fresh features arrive, stale ones are evicted, keeping the RAM usage of the dynamic window **bounded**.

(a) Experimental Result



(b) Edge Feature Proportion

Storage overhead ≤ **5.41%** of total edge feature storage (Figure (b))

(b) Theoretical Analysis

Theorem 4. *Given an L-layer T-GNN, the RAM window size (i.e., RAM usage for edge features) is bounded by* $\mathcal{O}(\alpha^L|V|)^9$.

Please refer to our manuscript for the proof.

Strengths

(a) Amortize the bursty startup I/O (disk-to-RAM) required to preload the entire graph across individual chunks.
(b) Decouple disk-to-RAM I/O for fresh and stale features, and overlap disk I/O with GPU computation.

Optimization 1: I/O-Efficient Feature Window Scheduling

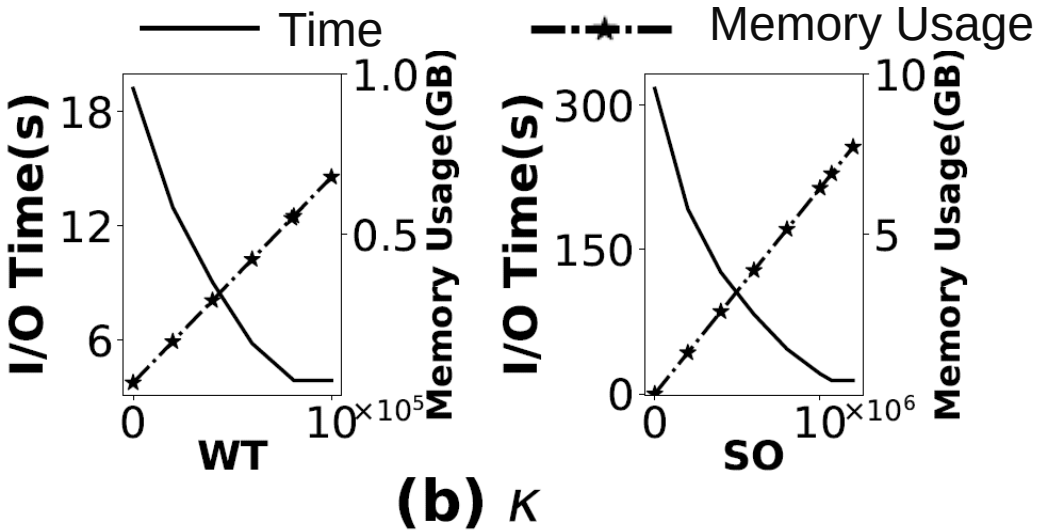
Other Optimizations

Idle Feature Scheduling

Motivation: If these idle features persist for long without reuse, they increase RAM usage without improving locality.

Method: We cap the number of idle features with a capacity parameter κ, applying an LRU eviction policy

We set κ dynamically based on the available RAM budget



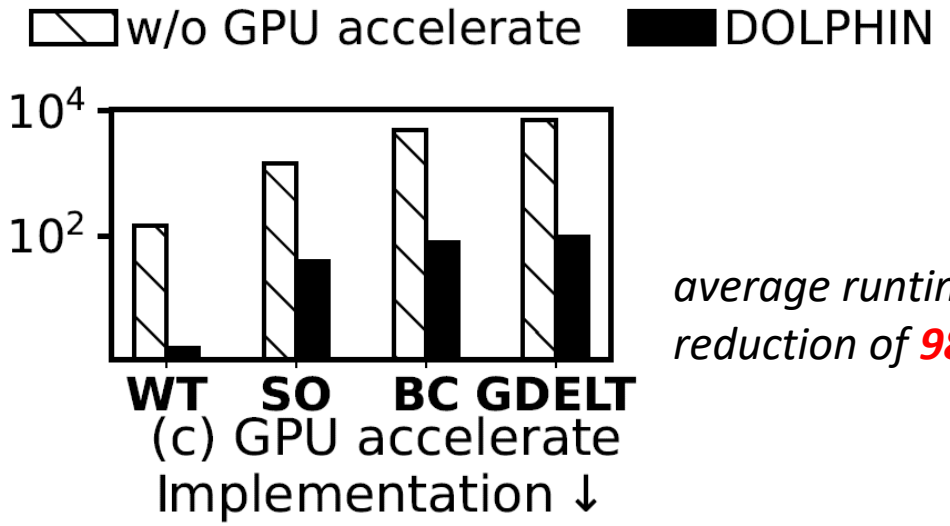
(b) κ

There is a trade-off between I/O time and RAM usage grows. (Figure (b))

GPU-based Stale Edge Identification

Motivation: Stale Edge Identification requires neighbor sampling for training nodes, which is **inefficient** with CPU multithreading.

Method: We design a GPU-based temporal sampler that significantly accelerates this process. (Figure (c))



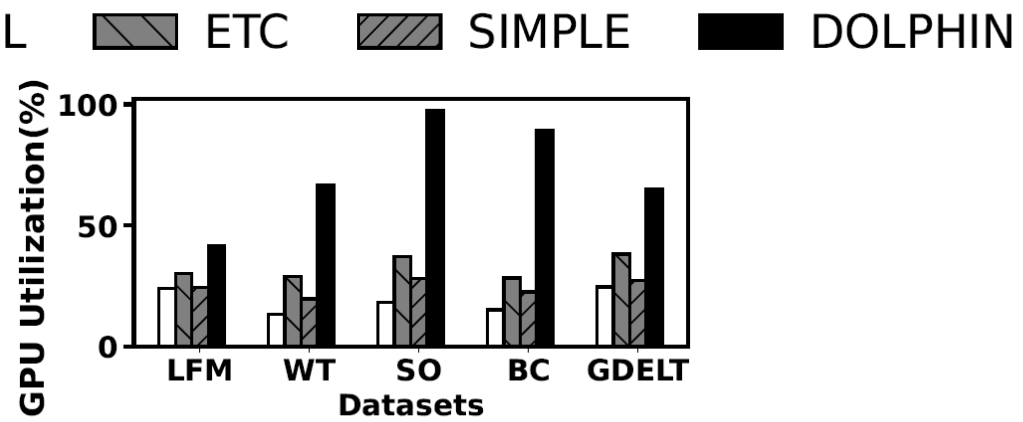
average runtime reduction of ***98.33 %***

Optimization 2: Dependency Graph-based Execution Planning

Motivation

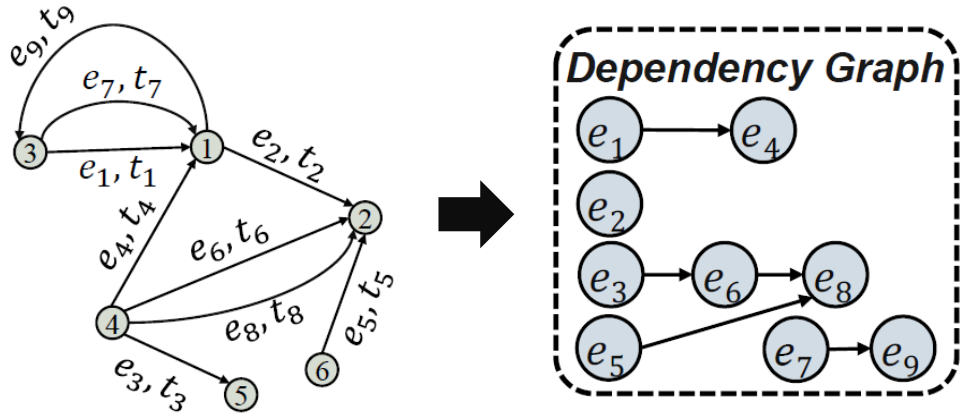
Existing systems typically overlook global temporal dependencies for batch construction

making it difficult to increase the batch size and resulting in GPU underutilization (Figures (b))



(b) GPU Utilization ↑

Dependency Graph: the edge-centric dual of the temporal graph



Dependency Graph Construction:

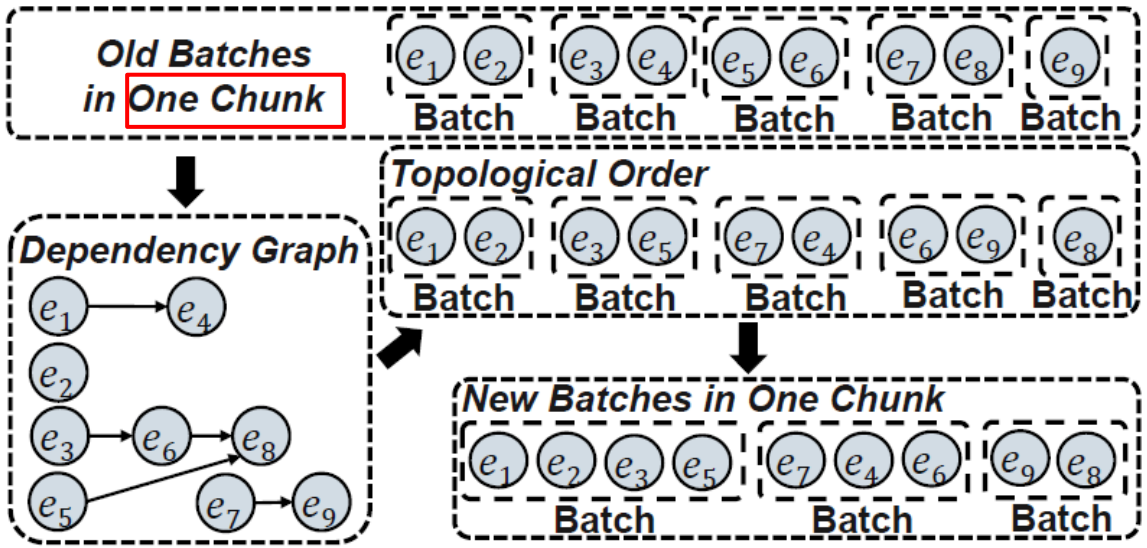
We iteratively identifies dependencies between edges in adjacent batches within a chunk.

We obtain a directed acyclic graph that naturally admits a topological order.

Please refer to Algorithm 1 in the manuscript for details.

Optimization 2: Dependency Graph-based Execution Planning

Parallel Execution Plan Computation



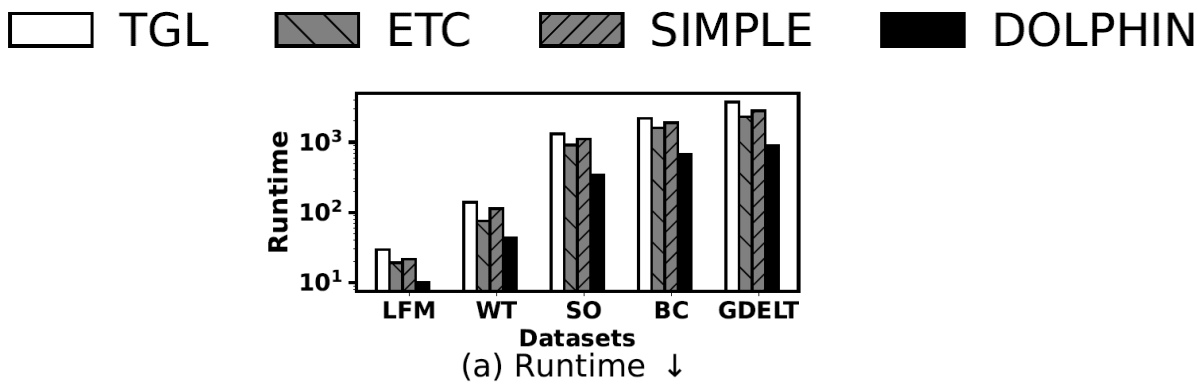
one chunk, one execution plan

Please refer to Algorithm 2 in the manuscript for details.

Due to temporal constraints, batches must be executed sequentially.

The overall runtime is scaling as O(k), where k is the number of batches

(a) Experimental Result



Improved runtime efficiency (Figure (a))

(b) Theoretical Analysis

Theorem 3. *DOLPHIN produces the minimal k for the parallelization plan design problem.*

Theorem 2. *Compared with the state-of-the-art RAM–GPU pipeline (ETC), DOLPHIN's Disk–RAM–GPU pipeline incurs lower bubble time [47], [48].*

Please refer to our manuscript for the proof.

Theorem 3 proves our algorithm achieves the minimal k.

Theorem 2 establishes the advantage of our system over the state-of-the-art system.

Strengths

(a) Balance dependency and concurrency by deriving an execution-plan with better parallelism without violating temporal-dependencies. (b) Enable plug-and-play integration with existing T-GNN systems.

Optimization 2: Dependency Graph-based Execution Planning

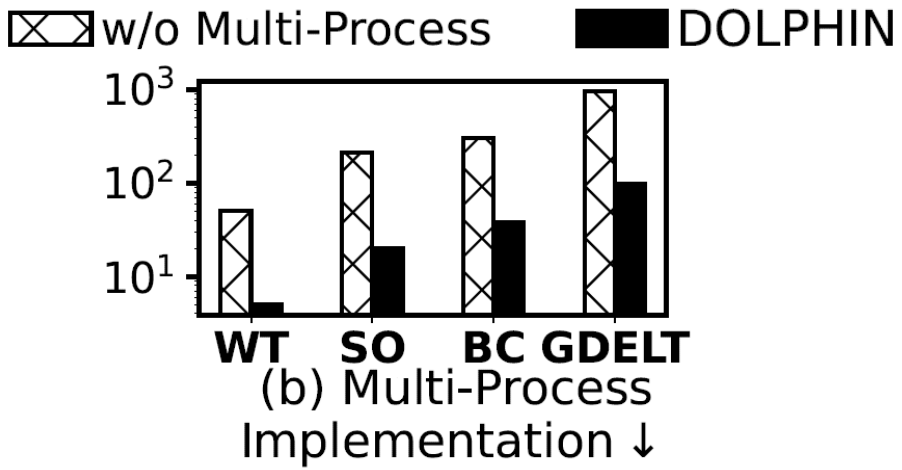
Other Optimizations

Multi-Process-based Planning

Motivation: Since each chunk has its own execution plan, we accelerate plan computation using multiple processes in parallel.

Method: We assign a dedicated process to each chunk

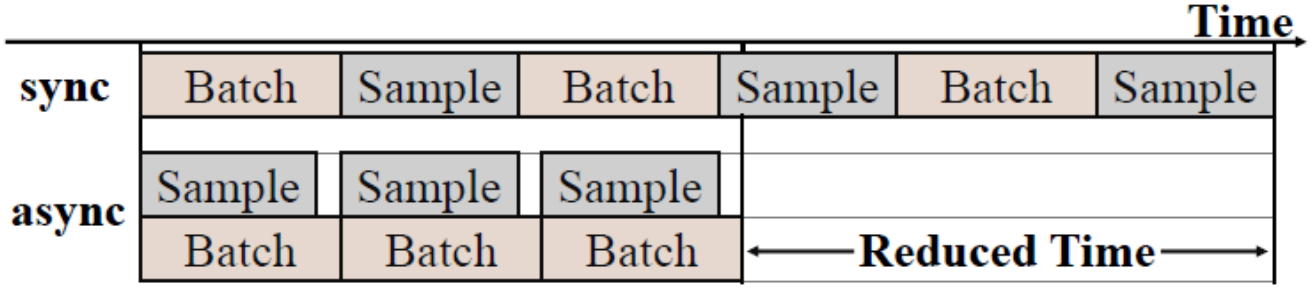
Time complexity is reduced from $O(|E|^2)$ to $O(\gamma^2)$, where $|E|$ is the total number of edges and γ is the number of edges per chunk.



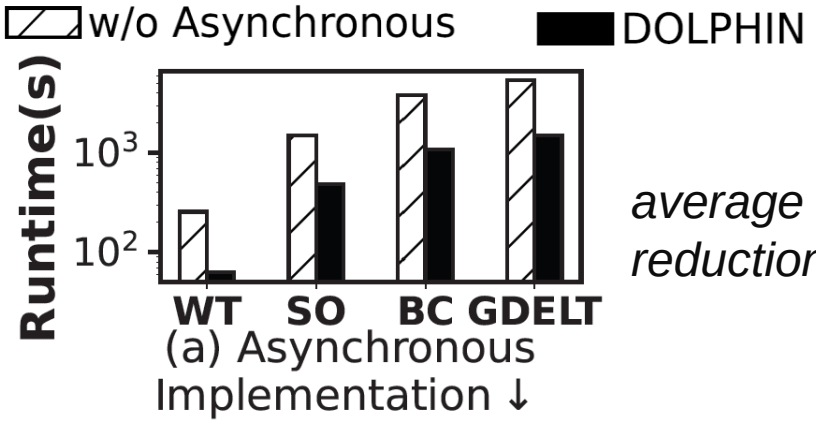
*average preprocessing time reduction of **88.27%***

Asynchronous Sampling and Training

Motivation: Training suffers from long-running synchronous operations, where sampling and training **execute sequentially**. This leaves GPUs idle while **waiting** for training to finish before sampling the next batch, as shown in the Figure below.



Method: We propose an in-GPU asynchronous sampling and training technique to further enhance efficiency.



*average runtime reduction of **71.75%***

Evaluations

Baseline Systems

TGL (VLDB 2022), ETC (VLDB 2024), SIMPLE (SIGMOD 2024), PipeTGL (VLDB 2025)

T-GNN Models

TGN, TGAT, TimeSGN

Six Real-world Graphs

Graph	$\|V\|$	$\|E\|$	$\mathcal{D}_v$	$\mathcal{D}_e$	Size (GB)		
					Node Feat	Edge Feat	Total
LastFM (LFM)	2K	1.3M	128	128	0.007	0.48	0.49
Wiki-Talk (WT)	1.1M	7.8M	172	172	0.73	5.01	5.74
Stack-Overflow (SO)	2.6M	63.4M	172	172	1.66	40.68	42.3
BITCOIN (BC)	24.5M	122.9M	172	172	15.7	78.7	94.5
GDELT	17K	191.3M	413	182	0.02	129.7	130
MAG	122M	1.3B	768	0	175	0	175

Runtime and Memory (For more information, please refer to our manuscipt)

TRAINING PERFORMANCE. T.: TIME PER EPOCH (S); M.: MAIN-MEMORY USAGE (GB); AP: AVERAGE PRECISION (%). BEST/SECOND-BEST RESULTS ARE BOLD/UNDERLINED. OOM: OUT OF MEMORY, WITH ELAPSED TIME IN PARENTHESES; TLE: >10 HOURS.

Env.		Frame.	LastFM			Wiki-Talk			Stack-Overflow			Bitcoin			GDELT		
			T. (s)	M. (GB)	AP (%)	T. (s)	M. (GB)	AP (%)	T. (s)	M. (GB)	AP (%)	T. (s)	M. (GB)	AP (%)	T. (s)	M. (GB)	AP (%)
TGN	$L=1$	TGL	29.40	3.78	83.02	139.92	11.52	94.43	1,304.92	57.18	88.39	2,175.81	155.00	90.85	5,710.92	171.95	98.31
		ETC	19.20	4.19	83.10	96.14	13.92	94.48	1,008.23	83.65	87.70	1,789.56	183.00	90.38	N/A	OOM(831)	N/A
		SIMPLE	17.51	3.51	82.70	112.93	11.81	94.54	1,885.19	62.98	88.05	3,292.27	158.00	90.87	2,895.80	175.12	98.28
		PipeTGL	24.02	8.2	82.53	224.12	17.24	93.91	1,476.53	102.48	88.19	2,074.28	176.49	91.02	4,776.51	186.79	98.21
		DOLPHIN	**10.08**	**3.42**	**83.15**	**63.00**	**8.46**	**94.89**	**542.02**	**29.54**	**88.62**	**1,079.82**	**110.85**	**91.56**	**1,489.20**	**19.91**	**98.38**
	$L=2$	TGL	353.59	4.31	86.23	1,313.52	11.94	95.08	12,699.67	59.00	88.83	8,556.42	155.00	90.31	TLE	TLE	N/A
		ETC	148.67	11.23	86.17	329.09	32.64	94.97	N/A	OOM(531)	N/A	N/A	OOM(835)	N/A	N/A	OOM(1,522)	N/A
		SIMPLE	100.08	3.72	86.14	369.91	12.82	94.92	4,645.92	83.55	89.06	N/A	OOM(1,035)	N/A	TLE	TLE	N/A
		PipeTGL	246.78	8.76	86.18	627.87	18.18	94.34	6,234.43	111.72	88.93	7,539.76	182.67	90.21	TLE	TLE	N/A
		DOLPHIN	**16.78**	**3.44**	**86.25**	**225.32**	**9.51**	**95.12**	**2,147.01**	**32.83**	**89.14**	**2,048.13**	**111.69**	**90.87**	**7,794.32**	**23.08**	**98.45**
TGAT	$L=1$	TGL	20.69	3.47	78.50	86.84	9.48	88.17	842.74	52.58	88.32	1,344.96	111.00	86.67	3,157.23	159.87	98.67
		ETC	11.66	4.24	78.18	63.75	11.95	88.05	503.79	78.75	88.17	929.71	139.00	86.54	N/A	OOM(738)	N/A
		SIMPLE	14.33	3.44	78.39	77.71	9.41	88.08	1,297.79	50.67	88.29	2,688.57	114.59	86.42	2,726.11	173.50	98.69
		PipeTGL	15.44	7.98	78.43	101.20	15.22	88.21	852.98	96.83	88.39	1,153.46	150.96	86.51	2,974.73	181.82	98.55
		DOLPHIN	**8.99**	**3.40**	**78.89**	**39.45**	**5.71**	**88.23**	**386.64**	**20.49**	**88.41**	**650.06**	**63.25**	**86.71**	**1,102.85**	**16.35**	**98.73**
	$L=2$	TGL	123.81	3.96	86.82	516.08	9.87	90.93	4,778.89	53.22	89.03	4,498.17	112.00	87.35	TLE	TLE	N/A
		ETC	75.85	11.84	86.71	243.44	29.21	90.83	N/A	OOM(506)	N/A	1,943.71	245.00	87.29	N/A	OOM(1,151)	N/A
		SIMPLE	55.86	3.54	86.90	245.21	9.53	90.87	4,904.71	66.08	88.52	4,621.54	117.81	87.39	TLE	TLE	N/A
		PipeTGL	105.64	8.20	86.89	350.31	15.94	90.58	4,097.80	97.62	88.78	3,538.59	154.78	87.32	TLE	TLE	N/A
		DOLPHIN	**12.89**	**3.39**	**87.32**	**119.92**	**6.25**	**90.96**	**1,434.83**	**23.82**	**89.35**	**1,613.28**	**63.69**	**87.44**	**5,561.48**	**19.37**	**98.79**
TimeSGN	$L=1$	TGL	30.31	3.68	81.92	179.19	11.52	92.46	1,405.14	59.09	84.50	2,266.30	155.00	88.94	5,646.19	175.60	98.75
		ETC	24.52	4.20	81.98	123.89	13.90	91.87	1,130.58	83.25	85.14	2,429.71	183.00	88.90	3,775.78	250.70	98.73
		SIMPLE	19.80	3.52	81.92	127.87	11.82	92.31	1,896.64	64.16	85.20	3,455.72	158.00	88.96	3,016.03	175.62	98.69
		PipeTGL	27.65	7.77	81.54	280.30	17.83	92.47	1,652.98	102.69	85.11	2,136.74	178.84	88.97	5,031.11	187.33	98.56
		DOLPHIN	**10.71**	**3.41**	**82.32**	**70.42**	**8.59**	**92.63**	**544.67**	**29.32**	**85.79**	**1,121.87**	**110.78**	**89.19**	**1,519.53**	**19.98**	**98.81**
	$L=2$	TGL	346.35	4.28	85.62	1,282.88	11.92	92.64	12,896.29	58.69	85.69	8,678.37	155.00	89.37	TLE	TLE	N/A
		ETC	132.13	11.60	84.65	343.07	32.31	92.38	N/A	OOM(501)	N/A	N/A	OOM(1,048)	N/A	N/A	OOM(1,220)	N/A
		SIMPLE	94.63	3.71	85.71	395.50	12.79	91.89	4,544.17	85.37	84.97	N/A	OOM(1,396)	N/A	TLE	TLE	N/A
		PipeTGL	236.76	8.78	85.82	624.80	18.89	92.88	6,216.59	103.40	85.60	7,726.95	183.93	89.34	TLE	TLE	N/A
		DOLPHIN	**18.32**	**3.39**	**85.86**	**202.99**	**9.18**	**93.09**	**2,072.68**	**33.59**	**85.73**	**1,892.19**	**111.74**	**89.50**	**7,141.27**	**21.60**	**98.83**

Up to 21.07x Improvement in Runtime Efficiency

Robust in Large Datasets

Up to 92.04% RAM Cost Reduction

Runtime and Memory (For more information, please refer to our manuscipt)

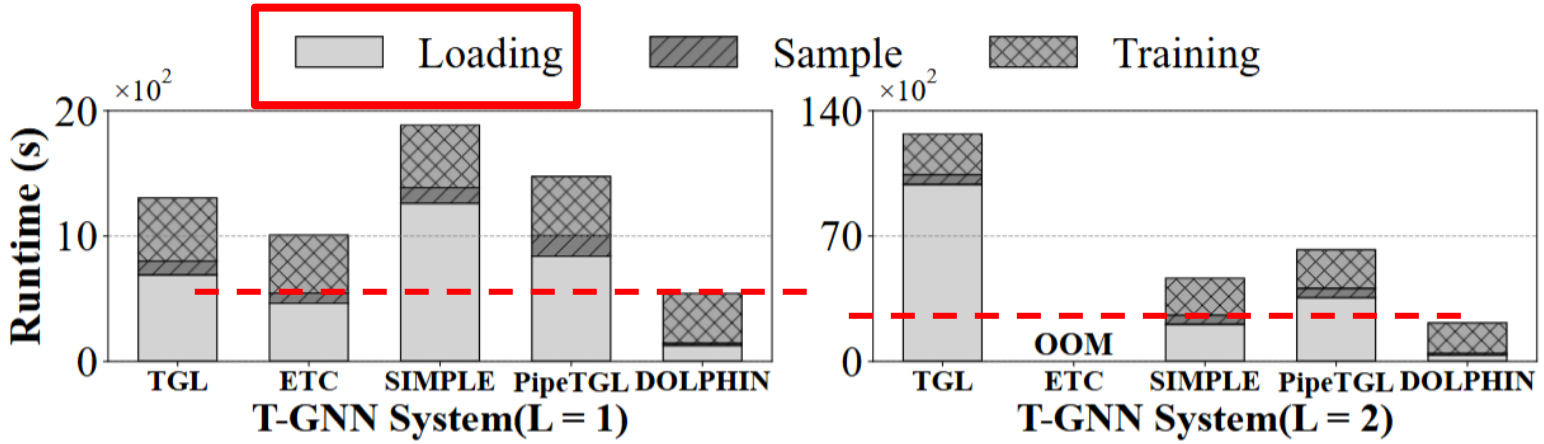


Fig. 8. 1 epoch in Stack-Overflow (A common legend is used for both subfigures.)

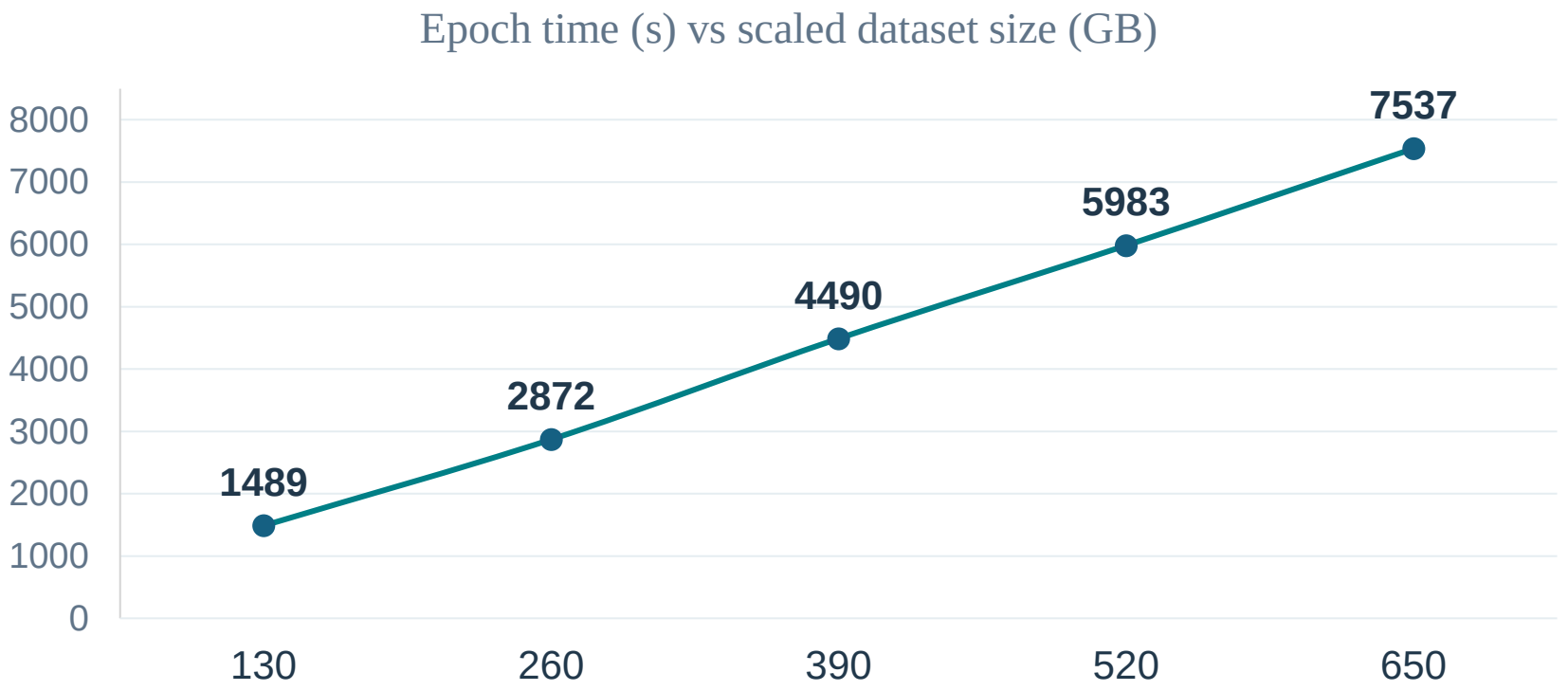
Accelerating Training by Addressing Data Loading Bottlenecks

TABLE IV
PREPROCESSING PERFORMANCE

L	Frame.	LastFM		Wiki-Talk		Stack-Overflow		Bitcoin		GDELT	
		Time. (s)	RAM (GB)	Time. (s)	RAM (GB)	Time. (s)	RAM (GB)	Time. (s)	RAM (GB)	Time. (s)	RAM (GB)
$L = 1$	SIMPLE	8.42	0.40	64.85	4.50	920.27	53.80	830.77	33.60	791.85	65.90
	PipeTGL	10.93	1.02	42.70	8.18	398.58	59.07	452.45	41.86	542.85	74.46
	DOLPHIN	**2.93**	**0.38**	**13.76**	**1.20**	**77.06**	**2.60**	**192.95**	**3.90**	**428.23**	**5.40**
$L = 2$	SIMPLE	25.92	2.00	162.75	15.60	3,180.60	307.00	1,307.44	56.20	3,278.43	234.20
	PipeTGL	10.53	1.11	43.78	8.23	405.96	59.09	493.84	41.93	553.77	74.51
	DOLPHIN	**2.93**	**0.38**	**13.76**	**1.20**	**77.06**	**2.60**	**192.95**	**3.90**	**428.23**	**5.40**

Low Preprocessing Overhead. Generate Once, Reuse Across Training Runs.

Training continues beyond RAM capacity



RAM: 256 GB

All tested baselines run out of memory from 256 GB onward

DOLPHIN successfully processes the GDELT workload scaled to 650 GB.

AP Performance (For more information, please refer to our manuscript)

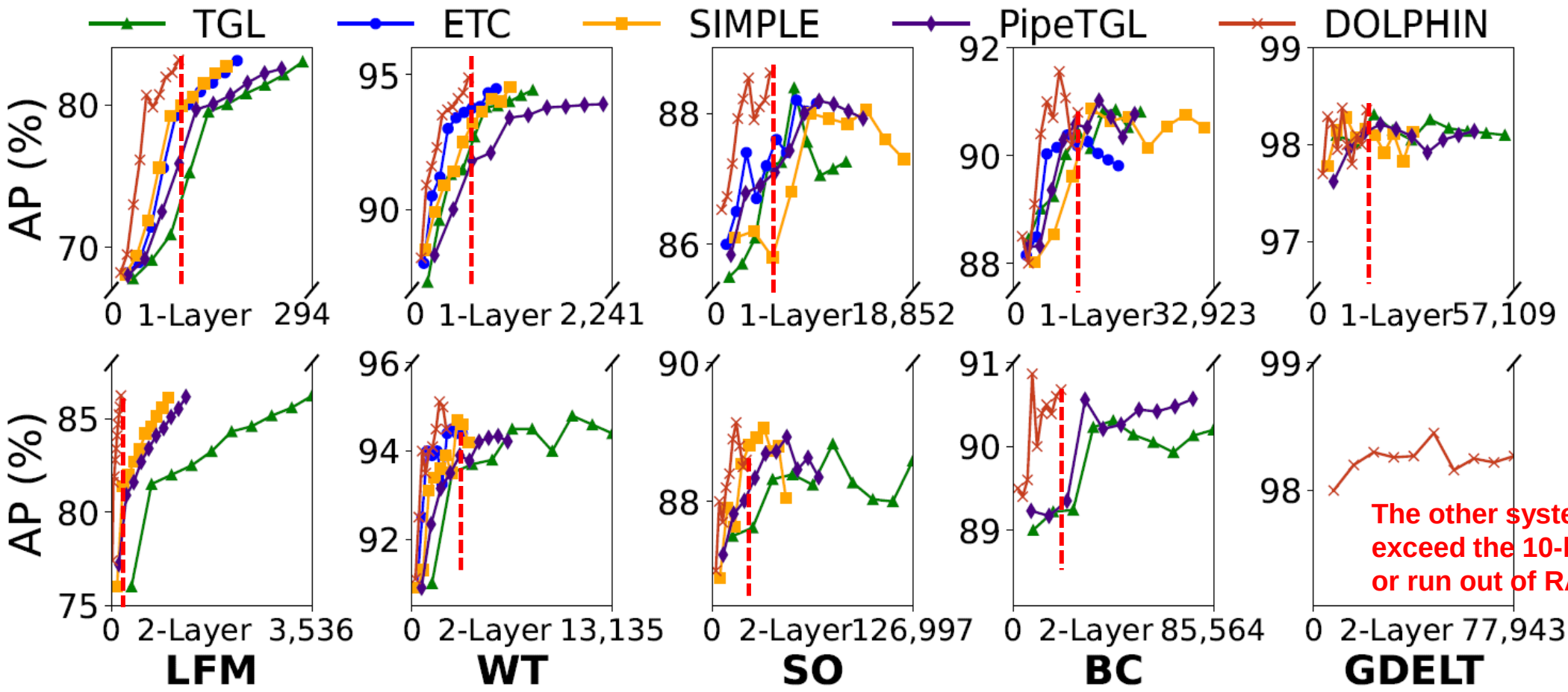


Fig. 11. AP vs. Runtime (s) (Training Time on the X-axis, Average Precision on the Y-axis. A common legend is used for all subfigures.)

Comparable AP with Faster Training (TGN, L = 1 and L = 2)

AP Performance (For more information, please refer to our manuscript)

Dataset	Best Baseline	DOLPHIN	Difference
LastFM	83.10%	**83.15%**	+0.05 pp
Wiki-Talk	94.54%	**94.89%**	+0.35 pp
Stack-Overflow	88.39%	**88.62%**	+0.23 pp
Bitcoin	91.02%	**91.56%**	+0.54 pp
GDELT	98.31%	**98.38%**	+0.07 pp

Comparable AP with Slight Improvements (TGN, L = 1)

Distributed Baselines

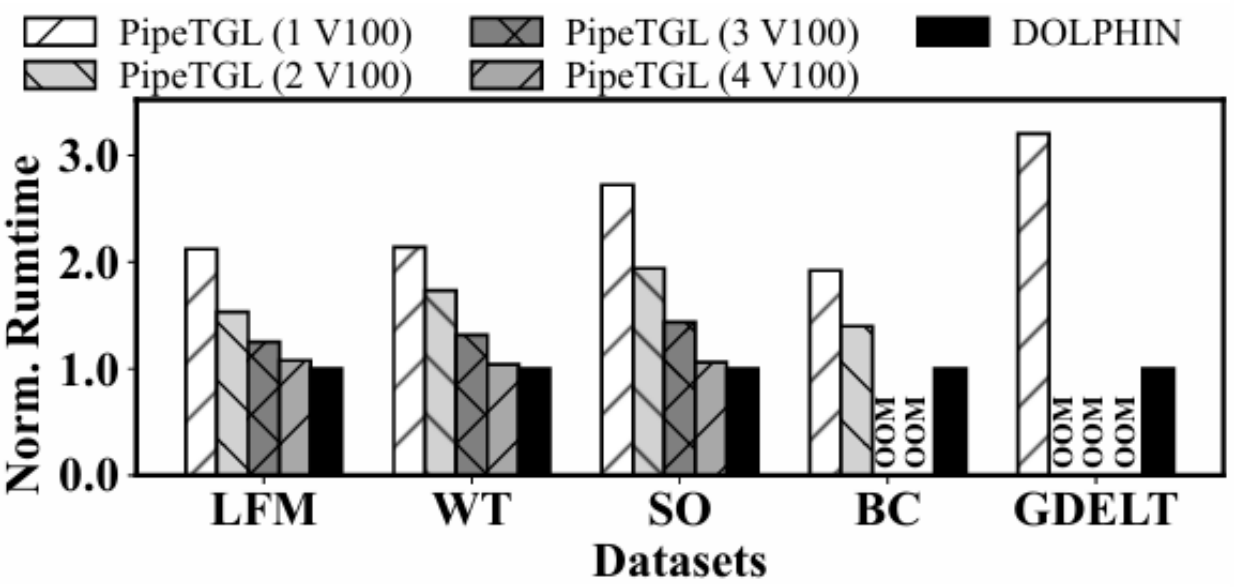


Fig. 15. PipeTGL vs. DOLPHIN

TABLE VIII
RUNTIME (SECONDS) AND RAM USAGE (GB) COMPARISON WITH DISTRIBUTED BASELINES ON THREE LARGE-SCALE TEMPORAL GRAPHS. THE BEST RESULTS ARE HIGHLIGHTED IN BOLD.

	LastFM		Stack-Overflow		GDELT	
Method	Time	RAM	Time	RAM	Time	RAM
DistTGL (4×V100)	14.12	6.93	646.71	83.62	2237.62	201.81
MSPipe (4×V100)	13.46	7.14	594.81	84.63	2086.54	205.81
MemShare (4×V100)	12.51	6.30	570.84	80.15	1835.06	198.73
DistTGL (1×V100)	29.16	3.72	1298.85	56.75	5705.83	171.58
MSPipe (1×V100)	28.66	3.81	1208.31	56.92	5305.86	172.03
MemShare (1×V100)	27.86	3.61	1176.81	54.31	5031.46	169.83
DOLPHIN (1×V100)	**10.08**	**3.42**	**542.02**	**29.54**	**1489.20**	**19.91**

Faster on a Single GPU than the Competing Method on Four GPUs

Different Hardware Configurations

TABLE X
PER-EPOCH TIME AND RAM COST PERFORMANCE ON GEFORCE RTX 4080 SERVER WITH 16GB GPU MEMORY AND 128GB RAM

Env.		Framework	LastFM		Wiki-Talk		Stack-Overflow		Bitcoin		GDELT	
			Time. (s)	$\mathcal{M}$. (GB)	Time. (s)	$\mathcal{M}$. (GB)	Time. (s)	$\mathcal{M}$. (GB)	Time. (s)	$\mathcal{M}$. (GB)	Time. (s)	$\mathcal{M}$. (GB)
TGN	$L = 1$	TGL	22.90	3.78	120.00	11.52	1,151.50	57.18	N/A	OOM(172)	N/A	OOM(24)
		ETC	14.48	4.19	82.60	13.92	881.00	83.65	N/A	OOM(506)	N/A	OOM(48)
		SIMPLE	13.81	3.51	78.90	11.81	1,508.00	62.98	N/A	OOM(92)	N/A	OOM(26)
		PipeTGL	17.49	8.20	105.74	17.24	1,236.43	108.67	N/A	OOM(92)	N/A	OOM(70)
		DOLPHIN	**7.87**	**3.42**	**37.42**	**8.46**	**365.88**	**29.54**	**743.82**	**110.85**	**1,141.13**	**19.91**
	$L = 2$	TGL	291.90	4.31	1,113.00	11.94	11,441.00	59.00	N/A	OOM(274)	N/A	OOM(26)
		ETC	75.36	11.23	235.50	32.64	N/A	OOM(227)	N/A	OOM(397)	N/A	OOM (47)
		SIMPLE	74.20	3.72	274.70	12.82	3,879.00	83.55	N/A	OOM(94)	N/A	OOM(28)
		PipeTGL	185.32	8.73	553.44	18.18	5,392.71	111.72	N/A	OOM(95)	N/A	OOM(71)
		DOLPHIN	**11.34**	**3.44**	**161.29**	**9.51**	**1,809.23**	**32.83**	**1,596.53**	**111.69**	**6,908.88**	**23.08**

Our method consistently delivers speedups and memory savings across different hardware configurations.

Ablation Study
(Model: TGN, L=1)

Component Enabled	Metric	Average reduction
Disk–RAM–GPU Pipeline	Runtime	83.75%
Feature Window Scheduling	RAM	55.92%
Dependency-aware Planning	Runtime	45.69%

Each comparison evaluates the effect of disabling a different component.

Conclusions

We propose DOLPHIN, a novel Chunk-Based Disk–RAM–GPU Pipelined Training mechanism to tackle the scalability challenges of T-GNN Training.

Key Optimizations

- I/O-Efficient Feature Window Scheduling → Addresses the I/O Bottleneck
- Dependency Graph-based Execution → Addresses compute bottleneck

State-of-the-art Performance

Experiments on six real-world graphs show DOLPHIN delivers up to **21.07×** faster training and reduces RAM usage by up to **92.04%** compared to state-of-the-art baselines., making it a practical solution for scalable dynamic graph learning.

Thank you!

References

[1] Yi et al. TGB-Seq Benchmark: Challenging Temporal GNNs with Complex Sequential Dynamics. ICLR 2025

[2] Chaurasia et al. Learning Enhanced Protein Representations from Molecular Dynamics Trajectories via Temporal GNNs. ICLR 2026

[3] Dileo et al. Applications of temporal graph learning for predicting the dynamics of biological systems. arXiv 2026

[4] Ding et al. WorldGraph: Graph-Native World Modeling. arXiv 2026