Very Large Data Bases
50th International Conference on Very Large Databases

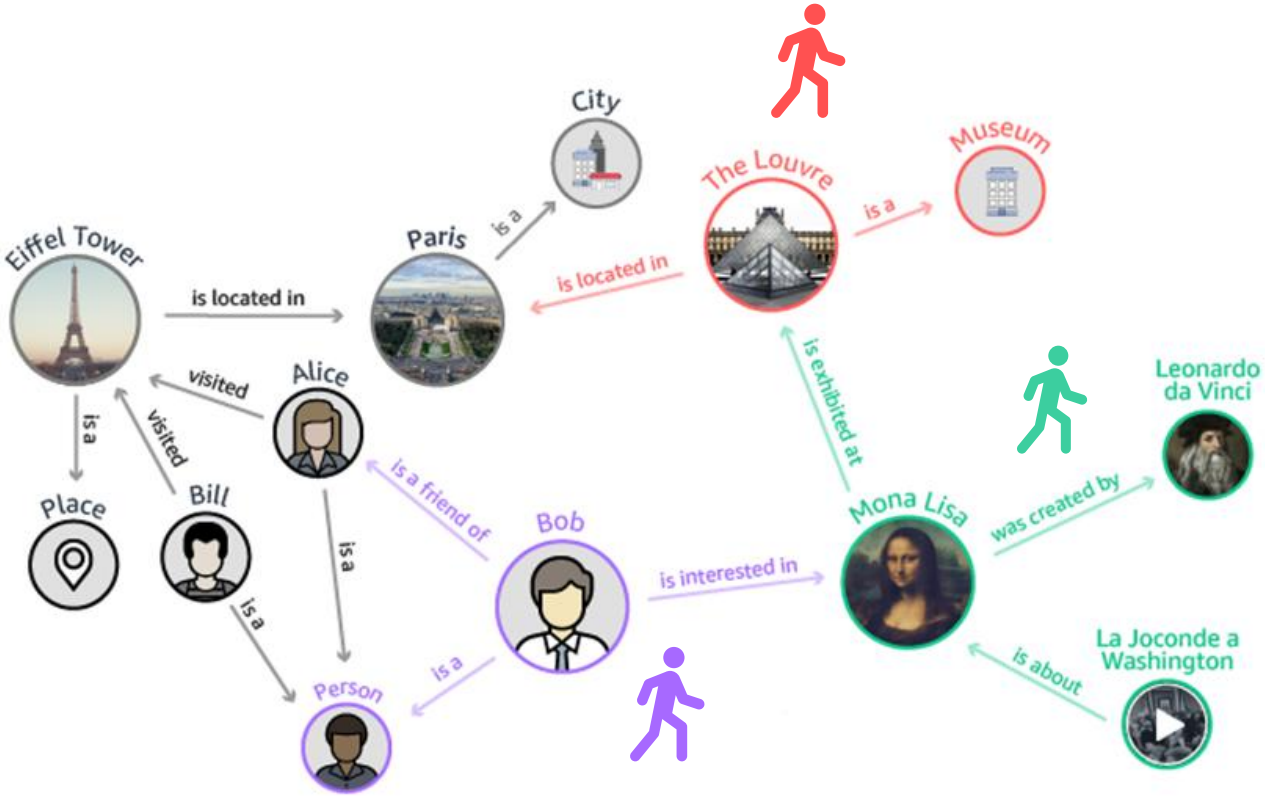
FlowWalker: A Memory-efficient and High-performance GPU-based Dynamic Graph Random Walk Framework

Junyi Mei[1], **Shixuan Sun**[1], Chao Li[1], Cheng Xu[1], Cheng Chen[2], Yibo Liu[1], Jing Wang[1], Cheng Zhao[2], Xiaofeng Hou[1], Minyi Guo[1], Bingsheng He[3], Xiaoliang Cong[2]

Shanghai Jiao Tong University[1]; ByteDance Inc[2]; National University of Singapore[3]

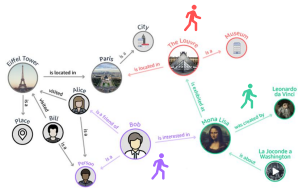
Graph Random Walk (RW)

- **Input**
  - Graph $\boldsymbol{G}$
  - A set of walkers $\boldsymbol{Q}$, with start vertices
- **Walking Process**
  - Each walker select a neighbor of current vertex at random
  - Move to the selected neighbor
  - Repeat until the termination condition is met
- **Output**
  - Walking path of walkers in $\boldsymbol{Q}$



Significance of Graph Random Walk

Graph random walk is the key operation to extract graph information, serving many downstream applications.



Graph Random Walk

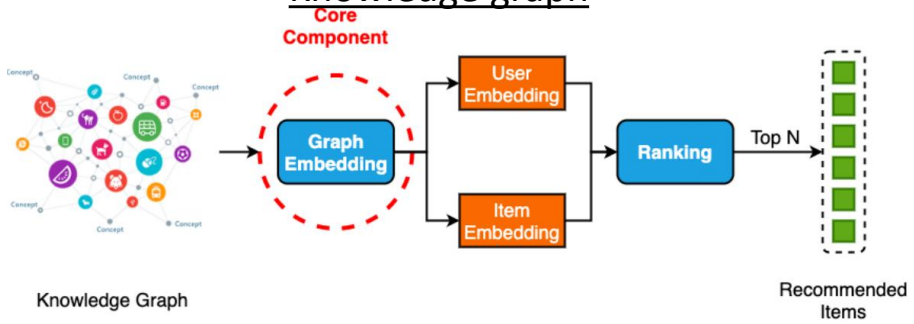
- **Graph neural network**
  - GNN, GCN, GRN
- **Graph ranking**
  - PPR, SimRank
- **Graph embedding**
  - DeepWalk, node2vec

Graph Information

Social network analysis

Recommendation system

Knowledge graph



Applications

Graph Random Walk Algorithms

- RW algorithms differ in the way to **select neighbor**

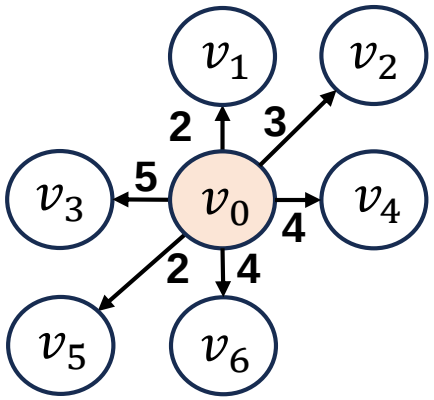
Unbiased: each neighbor has the same chance to be selected

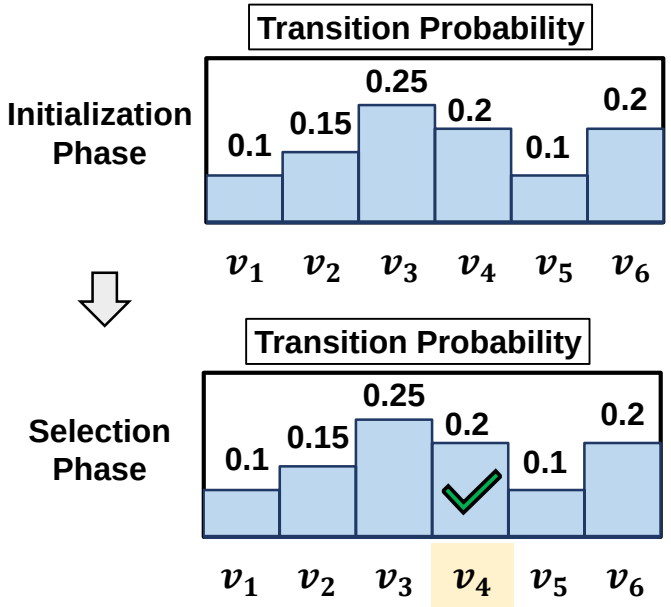
Biased

Static graph random walk (SGRW)
Fix transition probability (TP)

Dynamic graph random walk (DGRW)
TP is calculated during runtime

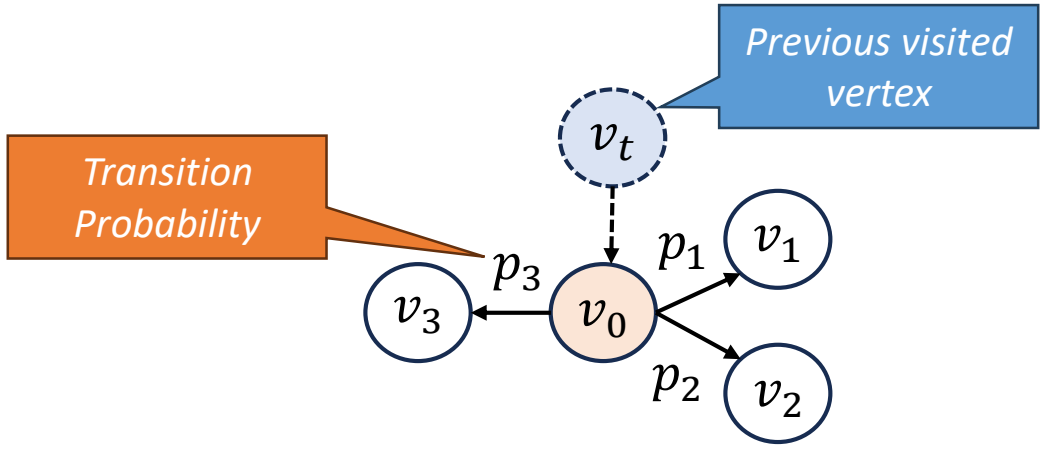
1~2页说明rw种类、介绍n2v和metapath，重新画图





Dynamic Graph Random Walk

- **Static Random Walk**
  - Calculate and store transition probability table in advance
  - Read the transition probability when needed
  - Suitable for the cases that transition probability stay fixed during runtime
- **Dynamic Random Walk**
  - Scan the edge and calculate transition probability during runtime
  - Suitable for the cases that transition probability may change



$$p(v_x) = weight(e_x) \times \begin{cases} \frac{1}{a}, & if\ dist(v_x, v_t) = 0 \\ 1, & if\ dist(v_x, v_t) = 1 \\ \frac{1}{a}, & if\ dist(v_x, v_t) = 2 \\ 0, & otherwise \end{cases}$$

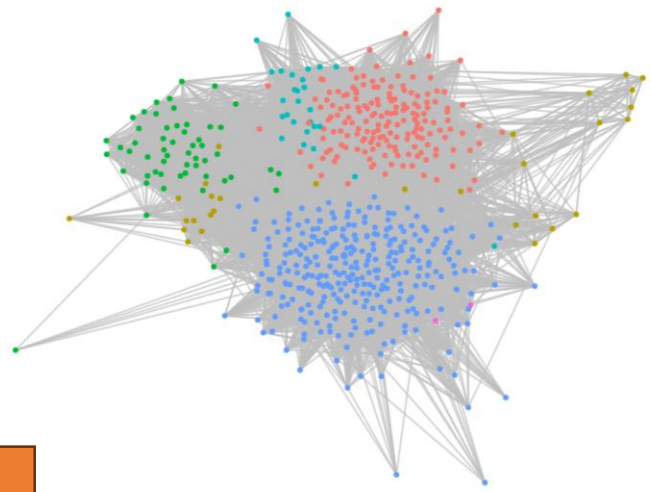
Node2Vec

p is dependent on runtime states v_t, cannot be computed in advance

Dynamic Sampling on GPU

Limited memory space

- Allocate $O(d_{max})$ memory buffer to store transition probability table
- Parallelism is restricted due to space cost



d denotes the vertex degree

The Twitter Graph [1]
$(d_{max} = 3 \times 10^6)$

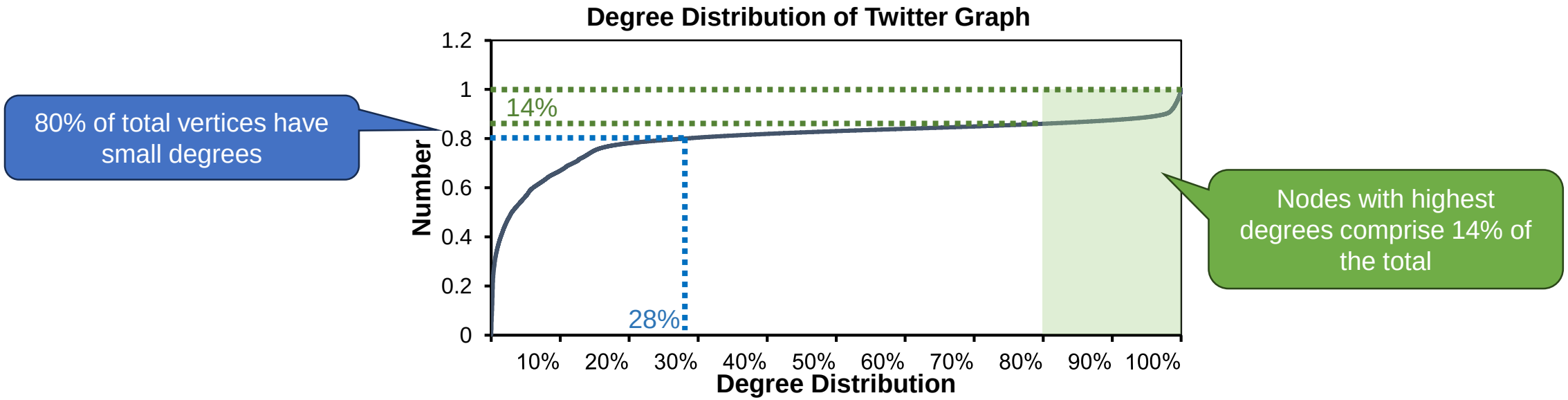
- ***A single query requires 11.45MB buffer***
- ***The number of query is large (i.e. 10^6)***
- ***A100 has only 40GB of DRAM***

[1]. https://ona-book.org/using-twitter.html

Dynamic Sampling on GPU

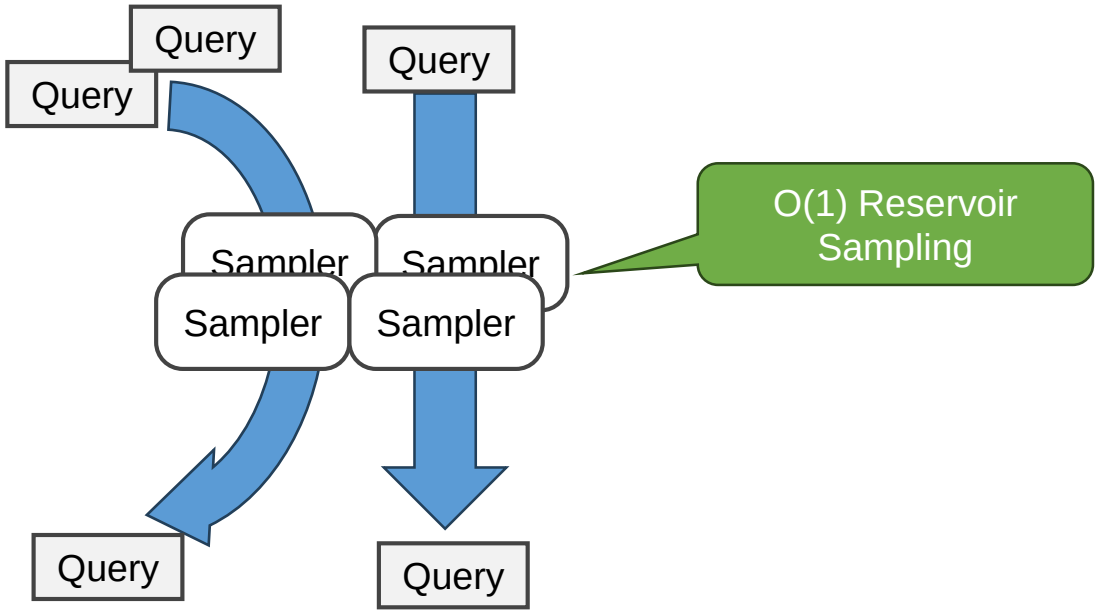
Load imbalance issues

- **Workload**
  - Workload is governed by vertex degree
  - Degrees in real-world graph follow power-law distribution
- **Hardware**
  - Huge amounts of computing cores exacerbates the imbalance problem



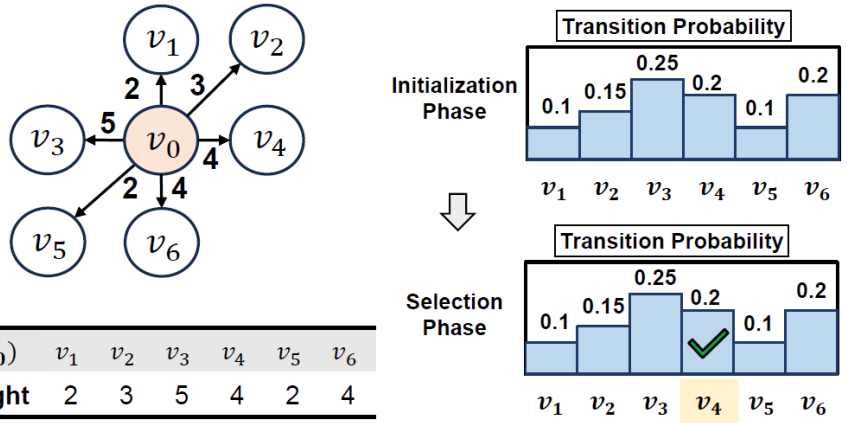
Our Solution

- FlowWalker: efficient dynamical walks at minimal memory cost
- Adopts **reservoir sampling** to reduce space complexity to $O(1)$
- High-performance processing engine, which leverages a **sampler-centric computation model**.



Reservoir Sampling

- **Sampling Methods**
  - Inverse Transform Sampling (ITS)
  - Alias Table Sampling (ALS)
  - Rejection Sampling (RJS)
- **Reservoir Sampling**
  - Pre-processing free
  - Diminishes space complexity to $O(1)$
  - High parallelism, easy adapt to GPU



$N(v_0)$	v_1	v_2	v_3	v_4	v_5	v_6
Weight	2	3	5	4	2	4

(a) An example graph. (b) Sampling a neighbor of v_0.

Sampling method	ITS	ALS	RJS	RS
Pre-processing	O(n)	O(n)	O(n)	-
Sampling	O(log n)	O(1)	O(1)~O(n)	O(n)
Space	O(n)	O(n)	O(1)	O(1)

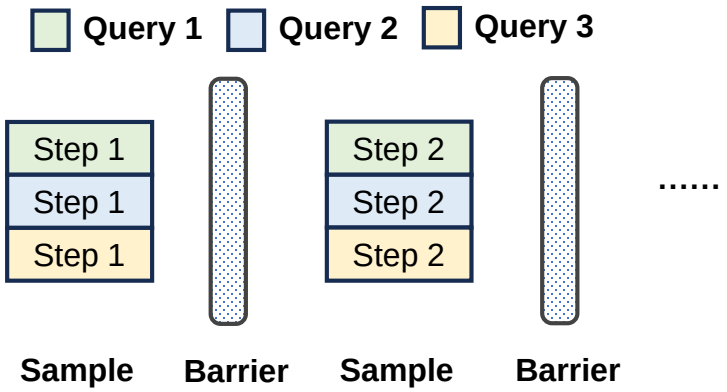
No pre-processing overhead

Accelerate through parallelization

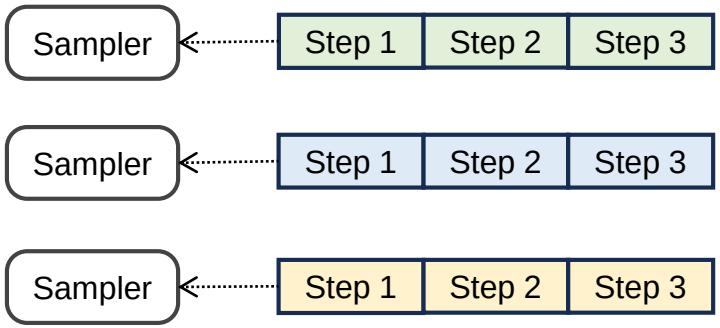
Minimal space cost

Execution Engine

- **Sampler-centric computation**
  - Organize threads into samplers
  - Each sampler is an independent worker; barrier free
  - A walking query will not be evicted until it completes
  - Fetch one task from the task pool when one query completes
- **Dynamic execution**
  - Instead of assigning tasks, samplers fetch tasks proactively



Step-centric



Sampler-centric

Experiment Setup

- **Baseline**
  - **Skywalker** [*PACT' 21*] – GPU-based framework
  - **LightRW** [*Sigmod' 23*] – FPGA-based dynamic RW framework
  - **ThunderRW** [*VLDB' 21*] – CPU in-memory framework
  - **DGL** – widely adopted GNN framework, run in dynamic mode
- **Environment**
  - Linux server with 256 GB of DRAM, and 31.5 GB/s of PCI-E bandwidth
  - One A100 (40 GB) GPU, 100 KB shared memory of each SM
  - One AMD Alveo U250 FPGA
  - One CPU with 16 cores and hyper-threading enabled
- **Datasets**
  - 10 read-world datasets, including **4 billion-scale datasets**
- **Applications**
  - DeepWalk, Personalized PageRank (PPR), Node2Vec, MetaPath

Dataset	Name	$\|V\|$	$\|E\|$	d_{max}	Size(GB)
com-youtube	YT	1.1 M	6 M	28K	0.05
cit-patents	CP	3.8 M	33 M	793	0.26
Livejournal	LJ	4.8 M	86 M	20K	0.66
Orkut	OK	3.1 M	234 M	33K	1.76
EU-2015	EU	11 M	522M	399K	3.93
Arabic-2005	AB	23 M	1.1B	576K	8.34
UK-2005	UK	39 M	1.6B	1.7M	11.82
Twitter	TW	42 M	2.4 B	3M	18.08
Friendster	FS	66 M	3.6 B	5K	27.16
SK-2005	SK	51 M	3.6 B	8.5M	27.16

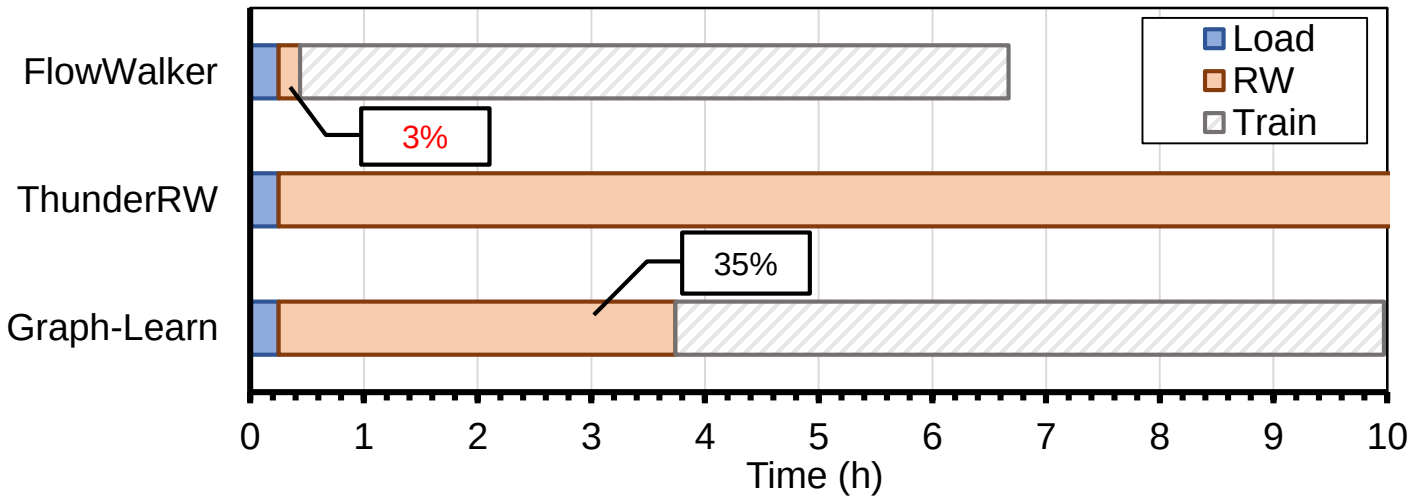
Datasets

Overall Comparison

- **FlowWalker is the only framework completing all test cases**
- **Time**
  - Up-to **92.2x** speedup to DGL (GPU), **315.8x** speedup to DGL (CPU)
  - Up-to **16.4x** speedup over LightRW (LRW)
  - Up-to **752.2x** speedup over ThunderRW (TRW)
  - Up-to **72.1x** speedup over Skywalker (SW)
- **Memory**
  - FlowWalker eliminates auxiliary data structures, reduce the space cost from O(n) to O(1)
  - The extra memory cost of FlowWalker is independent of graph size

Case Study

- Friend recommendation GNN in Douyin
- Test graph contains 227 million vertices and 2.71 billion edges
- FlowWalker reduces the RW time from **35%** (3.49 hours) to **3%** (13 minutes)



Training one epoch

Conclusion

- FlowWalker is a memory-efficient and high-performance GPU-based dynamic graph random walk framework.
- FlowWalker employs the reservoir sampling method.
- FlowWalker uses dynamic walking engine and sampler-centric model to enhance performance.
- FlowWalker samples graphs at a minimal memory cost while achieves significant performance improvements.

Source code available at github.com/junyimei/flowwalker-artifact

Contact: meijunyi@sjtu.edu.cn

Thanks!