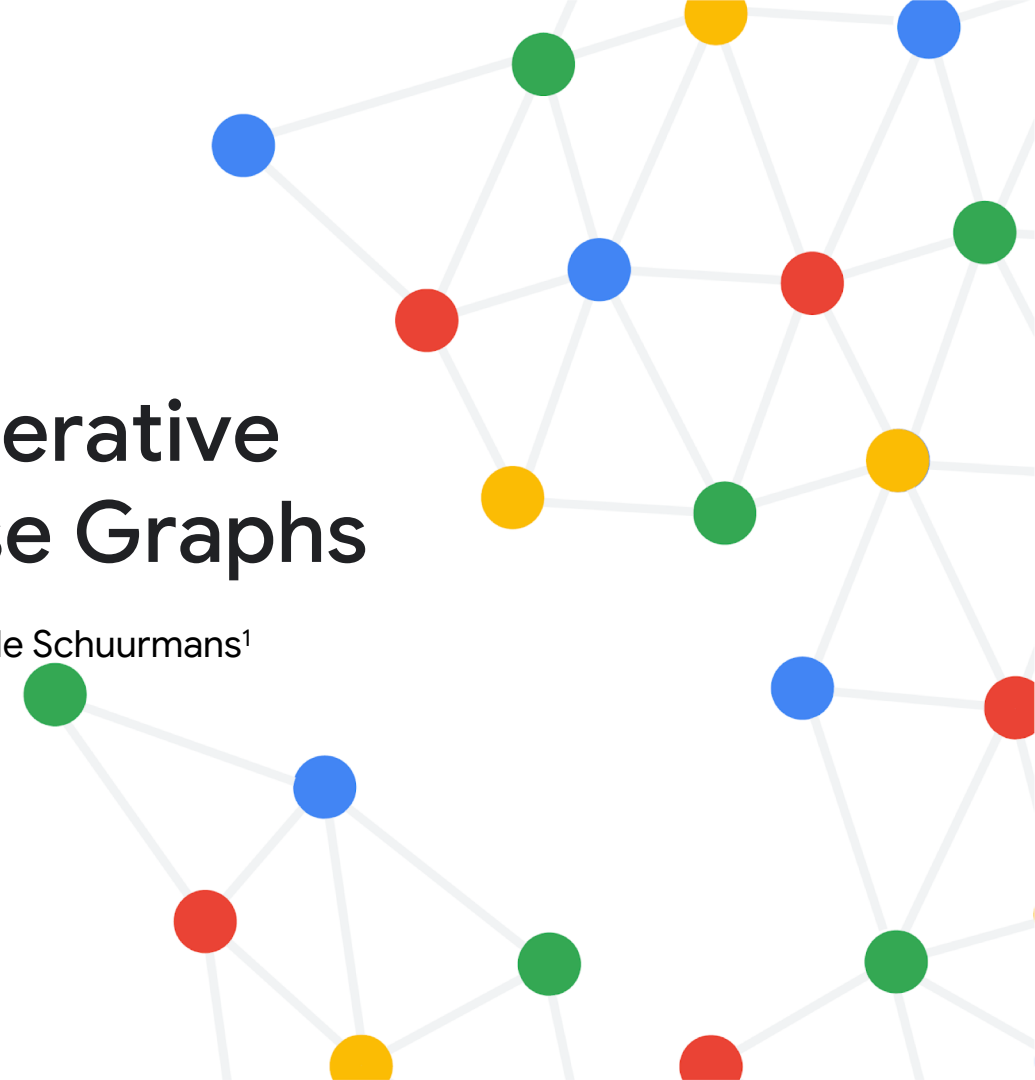
Scalable Deep Generative Modeling for Sparse Graphs

Hanjun Dai[1], Azade Nazi[1], Yujia Li[2], Bo Dai[1], Dale Schuurmans[1]

[1]Google Brain, [2]DeepMind

Google Research



Graph generative models

Given a set of graphs $\{G_1, G_2, ..., G_N\}$, fit a probabilistic model **p(G)** over graphs.

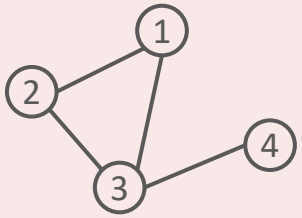
So that we can:

- sample from it to get new graphs: $G \sim p(G)$
- complete a graph given parts: $G_{rest} \sim p(G_{rest} \mid G_{part})$
- obtain graph representations

Can also be used for structured prediction $p(G|z)$.

Types of deep graph generative models

Leverage the sparse structure of graphs

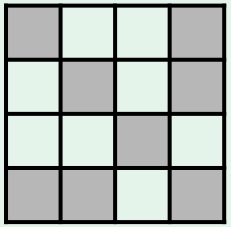


Junction-Tree VAE (Jin et al., 18)

NetGAN (Bojchevski et al., 18)

Deep GraphGen (Li et al., 18)

Modeling adjacency matrix directly
⇒ like an image



VAE (Kipf et al 16,
Simonovsky et al. 18)
Autoregressive (You et al., 18;
Liao et al., 19)

Google Research

Autoregressive graph generative models

Time complexity per graph during inference:

Model	Complexity (n nodes, m edges)	Scalability
Deep GraphGen (Li et al., 18)	$O((m + n)^2)$	~100 nodes
GraphRNN (You et al., 18)	$O(n^2)$	~2,000 nodes
GRAN (Liao et al., 19)	$O(n^2)$	~5,000 nodes
BiGG (Dai et al., 20)	$O((m + n) \log n)$	~100,000 nodes

This work

Or $O(n^2)$ for fully connected graph

Google Research

Autoregressive graph generative models

Time/memory complexity per graph during training:

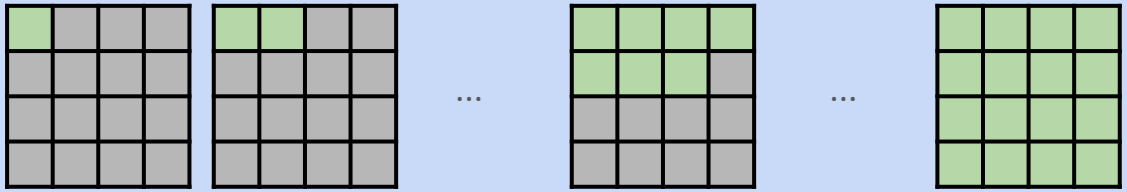
Model	# syncs during training	memory cost
Deep GraphGen (Li et al., 18)	O(m)	O(m(m+n))
GraphRNN (You et al., 18)	$O(n^2)$ or O(n)	$O(n^2)$
GRAN (Liao et al., 19)	O(n)	O(n(m+n))
BiGG (Dai et al., 20)	O(log n)	$O(\sqrt{m \log n})$

This work

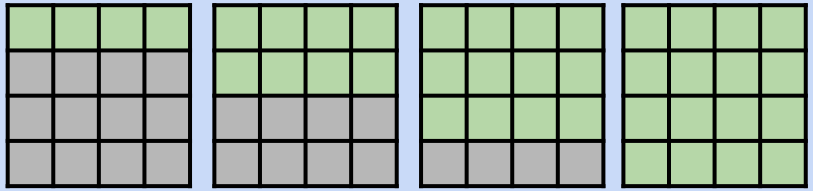
Google Research

Saving computation for sparse graphs

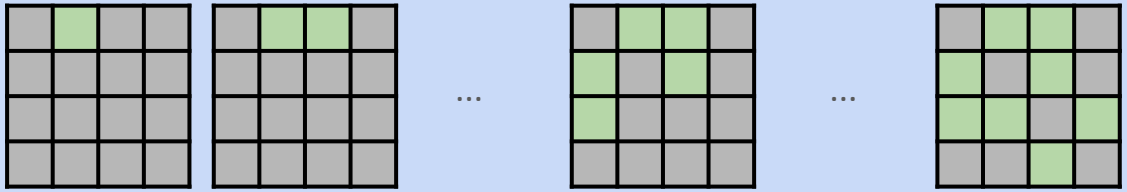
GraphRNN
$O(n^2)$



GRAN, GraphRNN-S
$O(n^2)$



BiGG (this work)
$O((m + n) \log n)$



Google Research

Autoregressive Generation of adjacency matrix

01 Generating one cell

02 Generating one row

03 Generating rows

Google Research

Autoregressive Generation of adjacency matrix

01 **Generating one cell**

02 Generating one row

03 Generating rows

Google Research

O(log n) procedure for generating one edge

Naive approach:

Given node **u**,
choose a neighbor **v**.

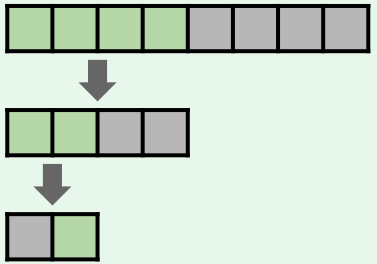
Choose 1 out of n using a softmax

O(n)

Efficient approach:

Recursively divide the range **[1, n]** into two halves, choose one.

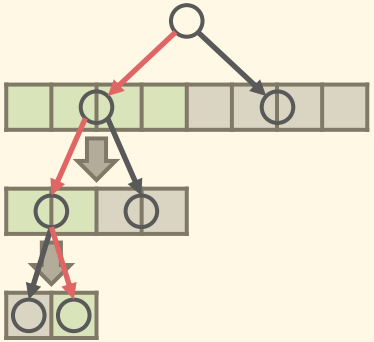
O(log n) decisions maximum



Google Research

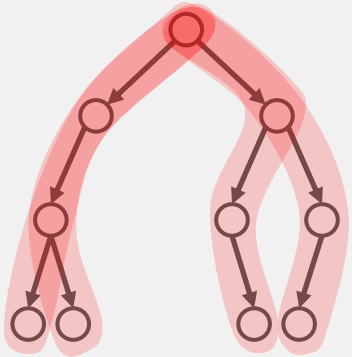
Binary tree generation

Following a path from root



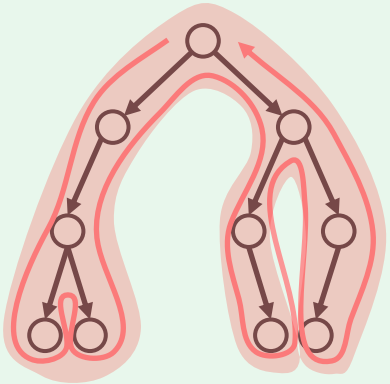
O(log n)

Generating neighbors separately?



$O(N_u \log n)$
N_u is the number of neighbors of node u

Generating via DFS



$O(|T|)$
$|T|$ is the tree size.
$|T| < \min\{N_u \log n, 2n\}$

Google Research

Autoregressive Generation of adjacency matrix

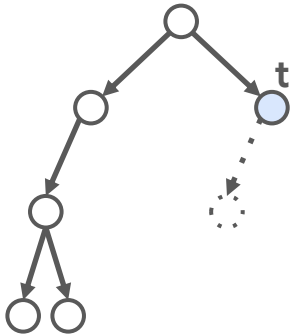
01 Generating one cell

02 **Generating one row**

03 Generating rows

Google Research

Autoregressive row-binary tree generation

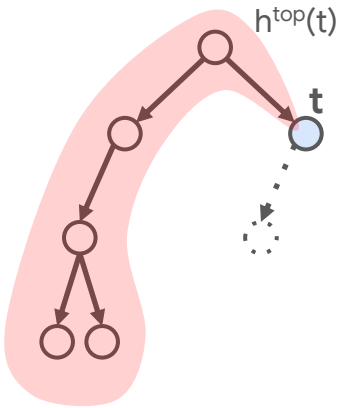


For node **t**, we first decide whether to generate left child.

```
def generate_tree(t):
  should generate left child?
```

Google Research

Autoregressive row-binary tree generation



For node **t**, we first decide whether to generate left child.

⇒**Generate left child:**

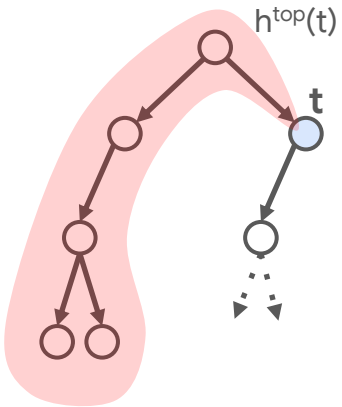
Conditioning on $\mathbf{h^{top}(t)}$, which summarizes existing tree (from top-down)

Has-left ~ Bernoulli(∘ | $h^{top}(t)$)

```
def generate_tree(t):
  should generate left child?
```

Google Research

Autoregressive row-binary tree generation



For node **t**, we first decide whether to generate left child.

⇒**Generate left child:**

Conditioning on $\mathbf{h^{top}(t)}$, which summarizes existing tree (from top-down)

↓

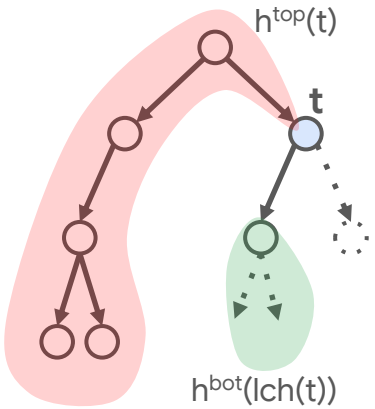
Has-left ~ Bernoulli(∘ | $h^{top}(t)$)

Yes? ⇒ Recursively generate left subtree

```
def generate_tree(t):
  should generate left child?
  if yes:
    create left child
    generate_tree(lch(t))
```

Google Research

Autoregressive row-binary tree generation



For node **t**, we first decide whether to generate left child.

⇒**Generate left child:**

Conditioning on $\mathbf{h^{top}(t)}$, which summarizes existing tree (from top-down)

Has-left ~ Bernoulli(∘ | $h^{top}(t)$)

↓

Yes? ⇒ Recursively generate left subtree

⇒**Generate right child:**

Conditioning on $h^{top}(t)$, and $\mathbf{h^{bot}(lch(t))}$, which summarizes the left subtree of **t** (from bottom-up)

Has-right ~ Bernoulli(∘ | $h^{top}(t)$, $h^{bot}(lch(t))$)

↓

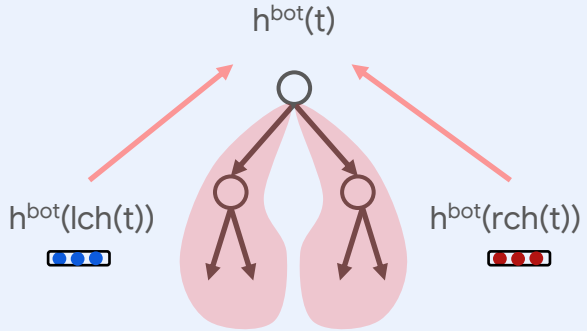
Yes? ⇒ Recursively generate right subtree

```
def generate_tree(t):
  should generate left child?
  if yes:
    create left child
    generate_tree(lch(t))
  should generate right child?
  if yes:
    create right child
    generate_tree(rch(t))
```

Google Research

Realize top-down and bottom-up recursion

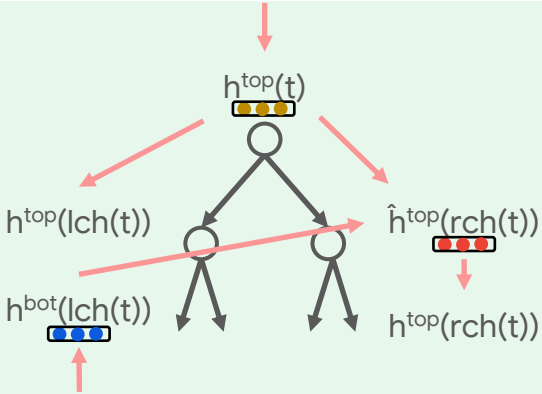
$h^{bot}(t) = \text{TreeLSTMCell}(\text{  }, \text{})$



$h^{top}(\text{lch}(t)) = \text{LSTMCell}(\text{  }, \overrightarrow{e_{left}})$

$\hat{h}^{top}(\text{rch}(t)) = \text{TreeLSTMCell}(\text{  }, \text{  })$

$h^{top}(\text{rch}(t)) = \text{LSTMCell}(\text{  }, \overrightarrow{e_{right}})$



Google Research

Autoregressive Generation of adjacency matrix

01 Generating one cell

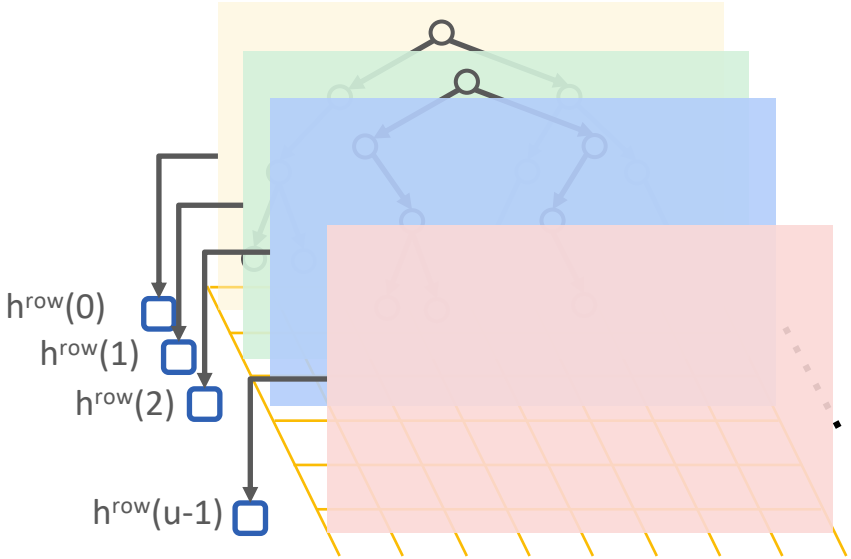
02 Generating one row

03 **Generating rows**

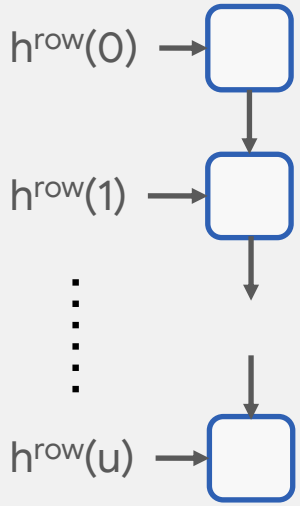
Google Research

Autoregressive conditioning between rows

To generate neighbors of node **u**, (i.e., **u**-th row)
How to summarize row_0 to row_{u-1}?



Use LSTM? – not efficient

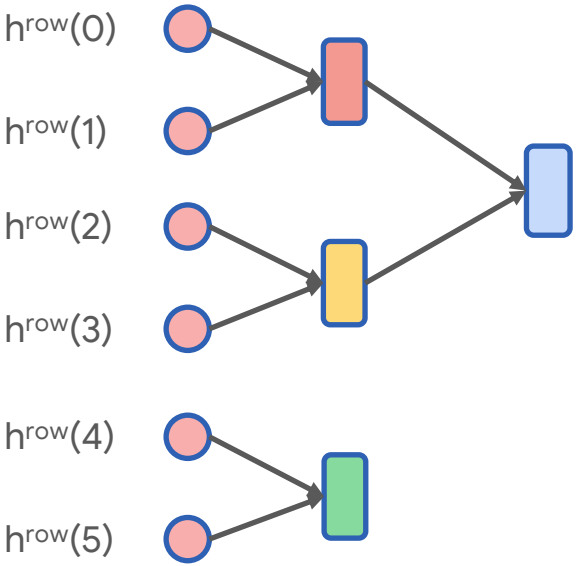


O(n) dependency length

Google Research

Fenwick tree for prefix summarization

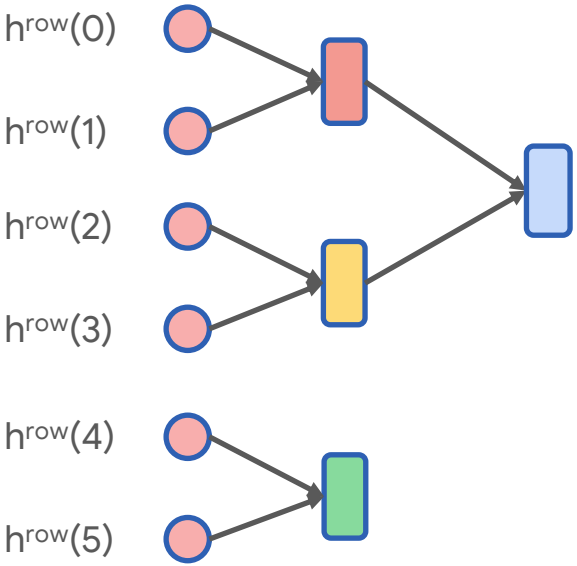
Fenwick tree: data structure that supports prefix sum and single modification



Google Research

Fenwick tree for prefix summarization

Fenwick tree: data structure that supports prefix sum and single modification



Obtaining “prefix sum” using low-bit query

Current row u	Required Context
u = 3	
u = 5	
u = 6	

At most O(log n) dependencies per row

Google Research

Optimizing BiGG

01 Training with O(log n) synchronizations

02 Model parallelism & sublinear memory cost

Google Research

Optimizing BiGG

01 **Training with O(log n) synchronizations**

02 Model parallelism & sublinear memory cost

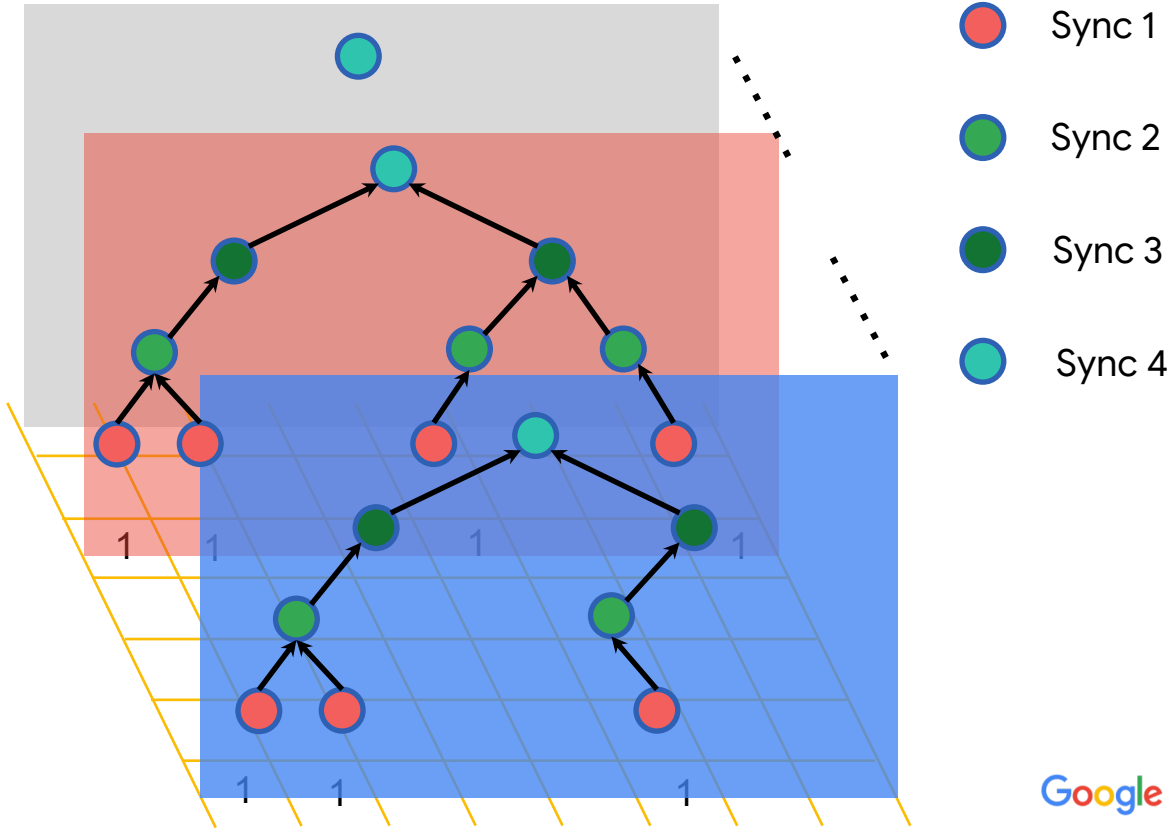
Google Research

Training with O(log n) synchronizations

Stage 1:

Compute all bottom-up summarizations for all rows

O(log n) steps



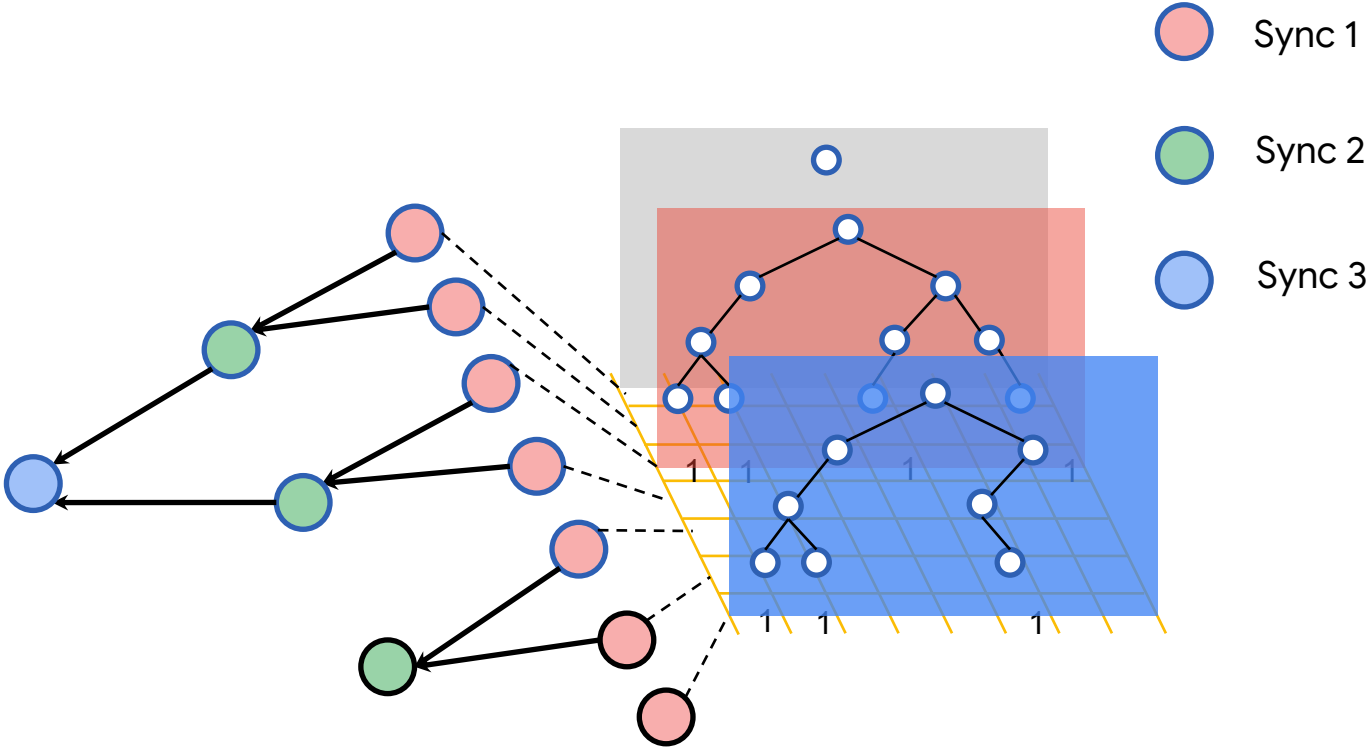
Google Research

Training with O(log n) synchronizations

Stage 2:

Construct entire Fenwick Tree

O(log n) steps



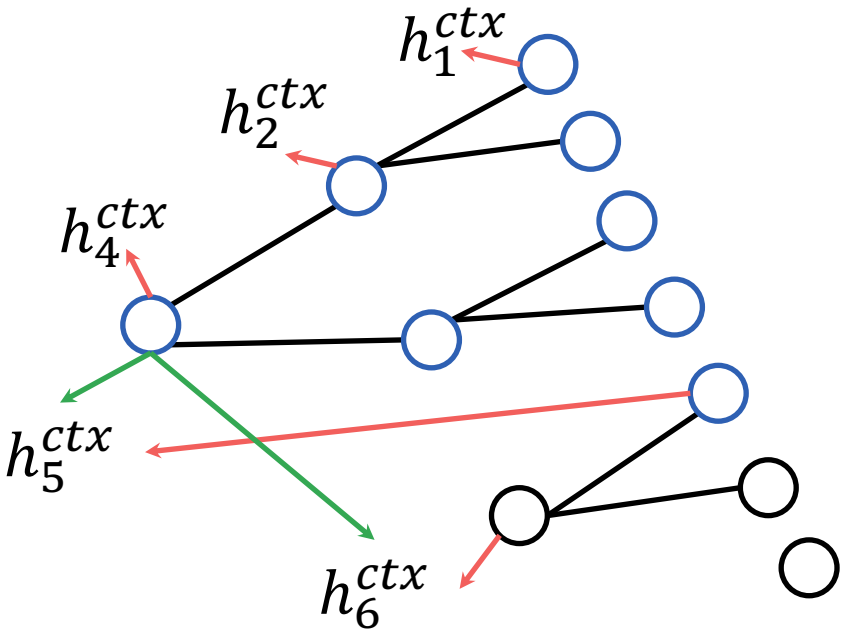
Google Research

Training with O(log n) synchronizations

Stage 3:

Retrieve all the prefix context

O(log n) steps



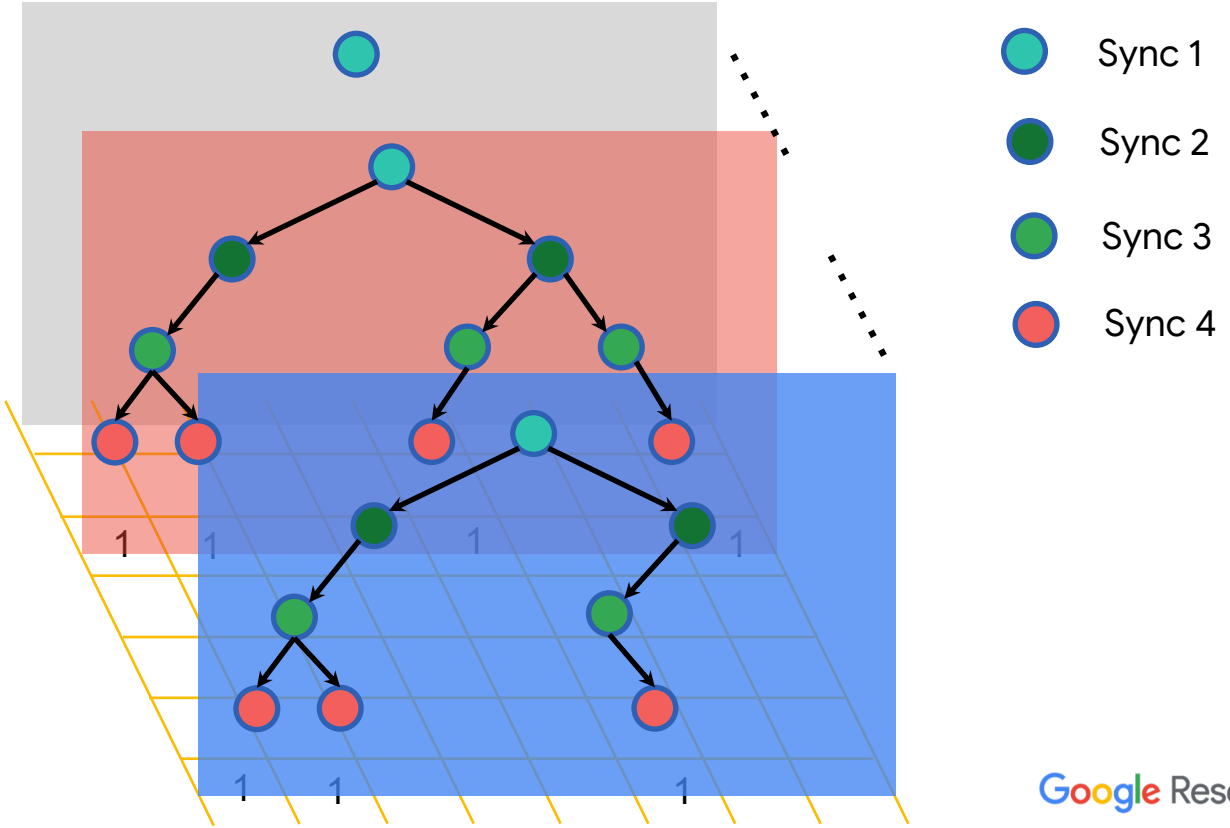
Google Research

Training with O(log n) synchronizations

Stage 4:

Compute
Cross-Entropy

O(log n) steps



Google Research

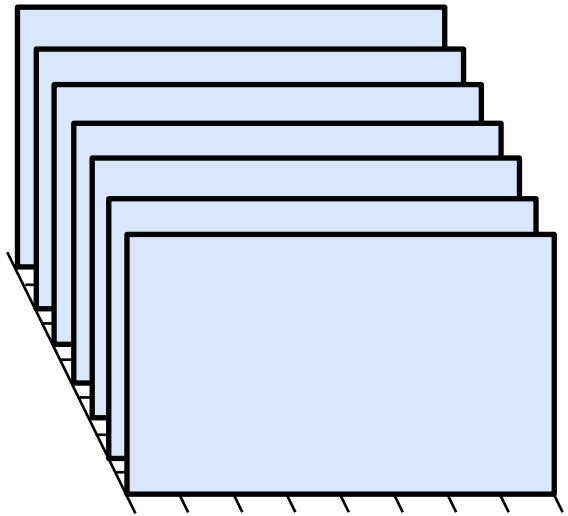
Optimizing BiGG

01 Training with O(log n) synchronizations

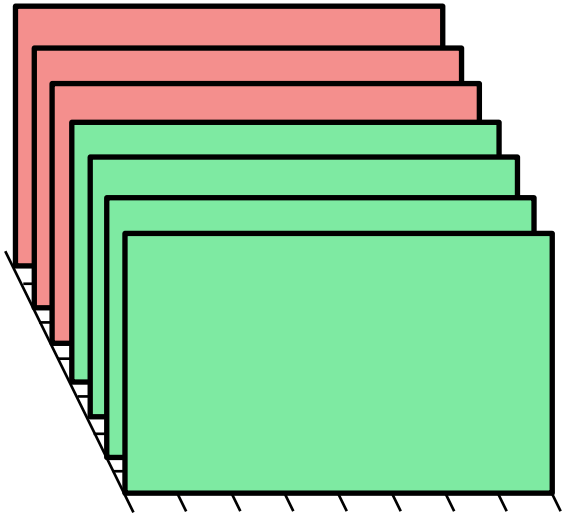
02 **Model parallelism & sublinear memory cost**

Google Research

Model parallelism

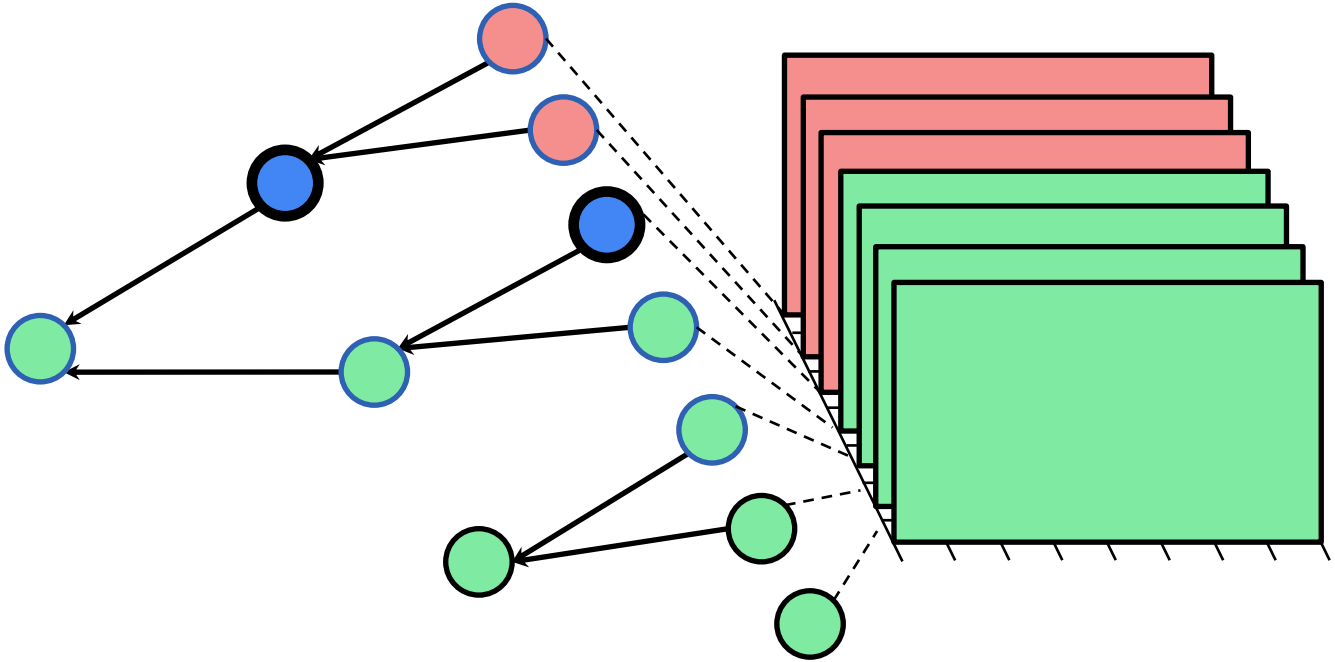


Model parallelism



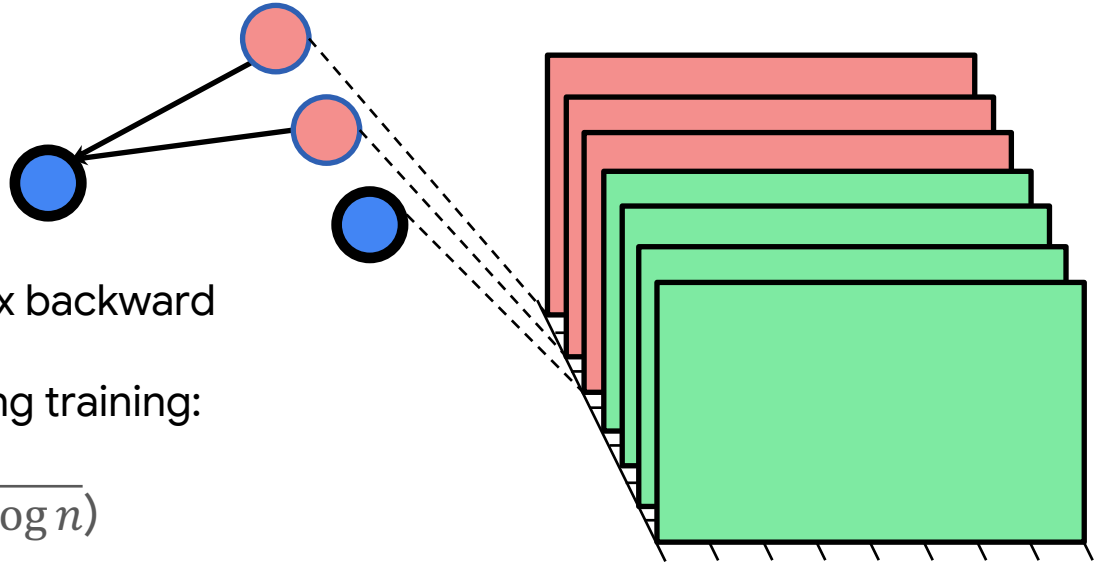
Google Research

Model parallelism



Google Research

Sublinear memory cost



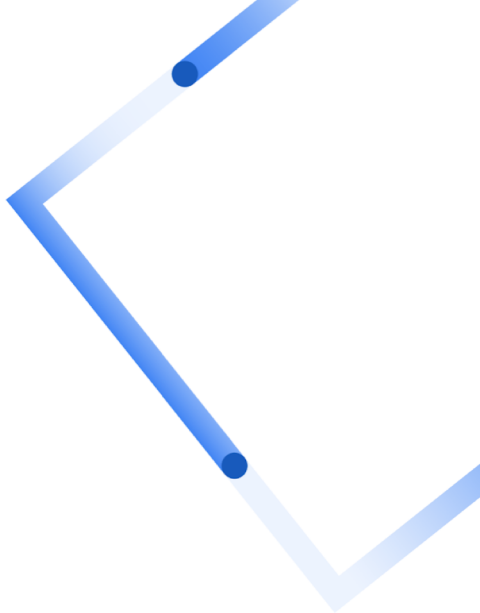
Run 2x forward + 1x backward

Memory cost during training:

$$O(\sqrt{m \log n})$$

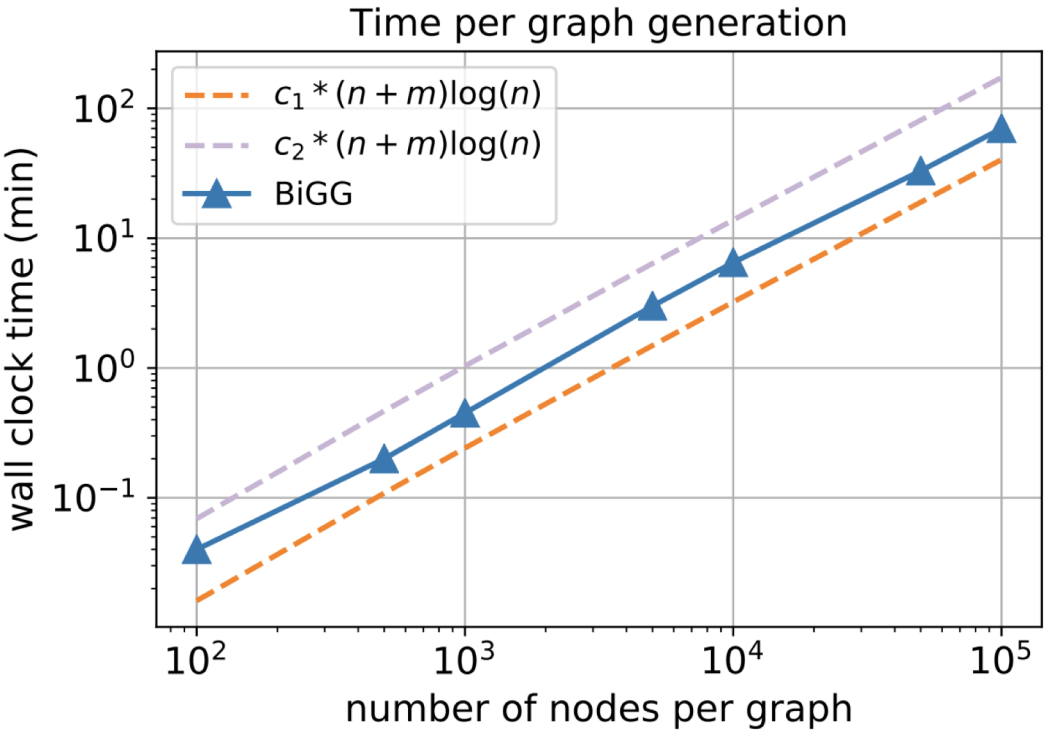
Google Research

Experiments



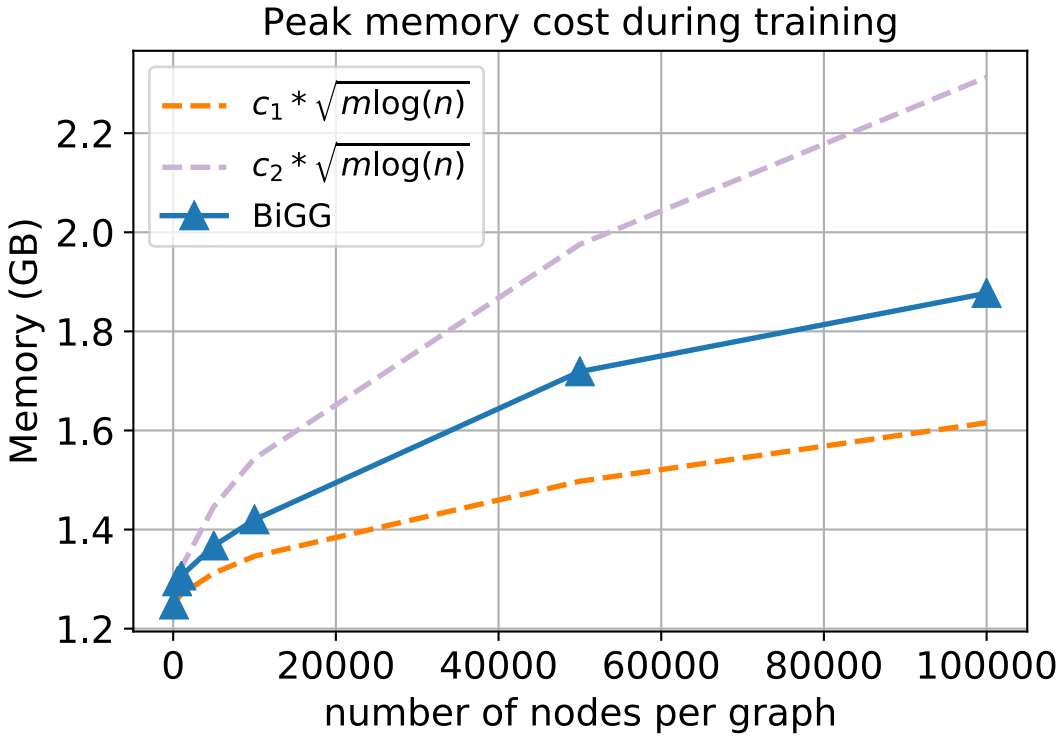
Google Research

Inference speed



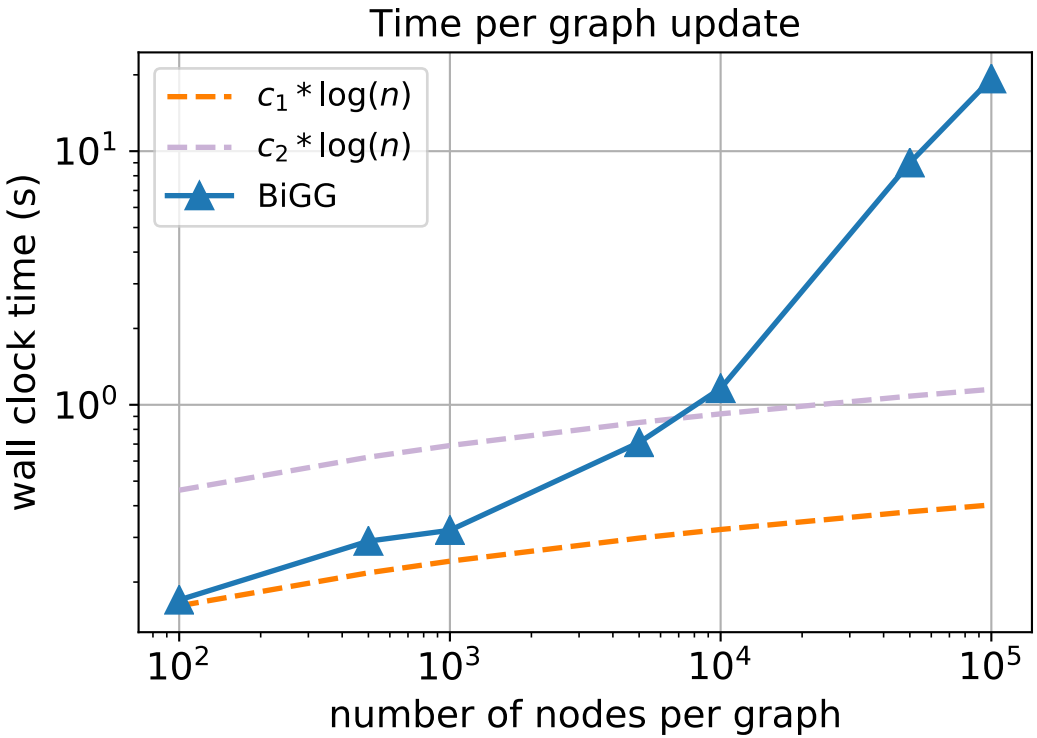
Google Research

Training memory



Google Research

Training time



Main reason: # GPU cores is limited

Google Research

Sample quality on benchmark datasets

Datasets		Methods					
		Erdos-Renyi	GraphVAE	GraphRNN-S	GraphRNN	GRAN	BiGG
Grid	Deg.	0.79	$7.07e^{-2}$	0.13	$1.12e^{-2}$	$8.23e^{-4}$	$\mathbf{3.60e^{-4}}$
	Clus.	2.00	$7.33e^{-2}$	$3.73e^{-2}$	$7.73e^{-5}$	$3.79e^{-3}$	$\mathbf{2.15e^{-5}}$
$\lvert V\rvert_{max} = 361, \lvert V\rvert_{avg} \approx 210$	Orbit	1.08	0.12	0.18	$1.03e^{-3}$	$1.59e^{-3}$	$\mathbf{4.43e^{-4}}$
$\lvert E\rvert_{max} = 684, \lvert E\rvert_{avg} \approx 392$	Spec.	0.68	$1.44e^{-2}$	0.19	$1.18e^{-2}$	$1.62e^{-2}$	$\mathbf{1.07e^{-2}}$
Protein	Deg.	$5.64e^{-2}$	0.48	$4.02e^{-2}$	$1.06e^{-2}$	$1.98e^{-3}$	$\mathbf{1.02e^{-3}}$
	Clus.	1.00	$7.14e^{-2}$	$4.79e^{-2}$	0.14	$4.86e^{-2}$	$\mathbf{2.24e^{-2}}$
$\lvert V\rvert_{max} = 500, \lvert V\rvert_{avg} \approx 1575$	Orbit	1.54	0.74	0.23	0.88	0.13	$\mathbf{2.73e^{-2}}$
$\lvert E\rvert_{max} = 258, \lvert E\rvert_{avg} \approx 646$	Spec.	$9.13e^{-2}$	0.11	0.21	$1.88e^{-2}$	$5.13e^{-3}$	$\mathbf{3.89e^{-3}}$
3D Point Cloud	Deg.	0.31	OOM	OOM	OOM	$1.75e^{-2}$	$\mathbf{2.34e^{-3}}$
	Clus.	1.22	OOM	OOM	OOM	0.51	**0.23**
$\lvert V\rvert_{max} = 5037, \lvert V\rvert_{avg} \approx 1377$	Orbit	1.27	OOM	OOM	OOM	0.21	$\mathbf{5.87e^{-3}}$
$\lvert E\rvert_{max} = 10886, \lvert E\rvert_{avg} \approx 3074$	Spec.	$4.26e^{-2}$	OOM	OOM	OOM	$7.45e^{-3}$	$\mathbf{5.11e^{-3}}$
Lobster	Deg.	0.24	$2.09e^{-2}$	$3.48e^{-3}$	$9.26e^{-5}$	$3.73e^{-2}$	$\mathbf{8.49e^{-5}}$
	Clus.	$3.82e^{-2}$	$7.97e^{-2}$	$4.30e^{-2}$	**0.00**	**0.00**	**0.00**
$\lvert V\rvert_{max} = 100, \lvert V\rvert_{avg} \approx 53$	Orbit	$2.42e^{-2}$	$1.43e^{-2}$	$2.48e^{-4}$	$2.19e^{-5}$	$7.67e^{-4}$	$\mathbf{7.43e^{-6}}$
$\lvert E\rvert_{max} = 99, \lvert E\rvert_{avg} \approx 52$	Spec.	0.33	$3.94e^{-2}$	$6.72e^{-2}$	$1.14e^{-2}$	$2.71e^{-2}$	$\mathbf{7.39e^{-3}}$
	Err.	1.00	0.91	1.00	**0.00**	0.12	**0.00**

Google Research

Sample quality as graph size grows

	0.5k	1k	5k	10k	50k	100k
Erdős–Rényi	0.84	0.86	0.91	0.93	0.95	0.95
GRAN	$2.95e^{-3}$	$1.18e^{-2}$	0.39	1.06	N/A	N/A
BiGG	$\mathbf{3.47e^{-4}}$	$\mathbf{7.94e^{-5}}$	$\mathbf{1.57e^{-6}}$	$\mathbf{6.39e^{-6}}$	$\mathbf{6.06e^{-4}}$	$\mathbf{2.54e^{-2}}$

Google Research

Summary

Advantages:

- Improve inference speed to $O(\min\{ (m + n) \log n, n^2\})$
- Enables parallelized training with sublinear memory cost
- Did not sacrifice the sample quality

Limitations:

- Limited by the parallelism of existing hardware
- Good capacity, but limited extrapolation ability

Google Research

Thank You

Hanjun Dai
Research Scientist
hadai@google.com

Google Research