Training Graph Neural Networks with 1000 Layers

3/4/2025

Arun Cheriakara Joseph,
Faculty Of Math

UNIVERSITY OF WATERLOO

Motivation

- Graphs Are Everywhere
- Real-World Examples
  - Netflix, Facebook, Amazon, Google Scholar
  - Internet = Vast Web Graph

Why Graph Neural Networks (GNNs)?

- Designed for graph-structured data
- Message passing between nodes
- Impressive results on **small** graphs

The Scaling Problem

- Memory bottleneck for large graphs
- Deep GNNs → More parameters → More memory
- Sampling/partitioning exist but not always ideal

Inspiration from NLP & Efficient Architectures

- NLP success: GPT, BERT, GPT-3 (scale improves performance)
- Efficient methods in NLP:
  - Reversible networks (RevNets)
  - Weight tying & equilibrium models

Proposed Methods

- Reversible Connections
- Weight-Tied Networks
- Graph Equilibrium (Implicit) GNN

Setup

- Input: A graph **G = (V, E)**
- N=|V|, M=|E|
- Adjacency matrix $A \in \mathbb{R}^{N \times N}$
- Node feature matrix X $\in \mathbb{R}^{N \times D}$
- (Optional) Edge feature matrix U $\in \mathbb{R}^{M \times F}$
- GNN operator: $f_w(X, A, U) \rightarrow X'$

Over-parameterized GNNs

- Deep GNNs often use **residual connections**
  - $X' = f_w(X, A, U) + X$
- Memory cost for L-layer GNN: $O(LND)$
  - L is the number of GNN layers,
  - N is the number of vertices,
  - D is the size of vertex features.
- Activation storage > parameter storage

Grouped Reversible GNN

- Inspired by **RevNets** + grouped convolutions
- Split features into C groups
- Reversible updates → No storing intermediate states
- Memory complexity → $O(ND)$

UNIVERSITY OF WATERLOO

Forward Pass in Grouped Reversible GNNs

- Divide input vertex feature matrix X into C groups.
- Compute an exchange term
- Process each group iteratively

$$X'_0 = \sum_{i=2}^{C} X_i \qquad (2)$$

$$X'_i = f_{w_i}(X'_{i-1}, A, U) + X_i,\ i \in \{1, \cdots, C\}, \qquad (3)$$

UNIVERSITY OF WATERLOO

Efficient Backpropagation

- Recompute X_i
- Recompute X_0'
- Reconstruct X_1

$$X_i = X_i' - f_{w_i}(X_{i-1}', A, U), \ i \in \{2, \cdots, C\} \tag{4}$$

$$X_0' = \sum_{i=2}^{C} X_i \tag{5}$$

$$X_1 = X_1' - f_{w_1}(X_0', A, U). \tag{6}$$

Avoiding Dropout Issues

- Dropout adds randomness, making reconstruction difficult.
- Solution: Use **shared dropout patterns** across all layers.
- This prevents memory from scaling with depth.

$$\mathit{Memory\ Complexity}:\ O(ND)$$

Memory Complexity & Practical Benefits

- Activations: $O(ND)$, not $O(LND)$
- Works with GCN, GAT, etc.
- Enables 1000+ layers on one GPU

Weight-Tied GNNs: The Idea

- Instead of learning different weights for each layer, we **reuse the same function** across layers.
- Reduces parameter count, making the model more **memory-efficient** and **generalizable**.

$$f_w^{(1)} := f_w^{(2)} \ldots := f_w^{(L)}, \tag{9}$$

Two Types of Weight-Tied GNNs

- **Weight-Tied Residual GNNs**:
    1. Shares weights across layers.
    2. **Still requires storing activations**, so memory complexity remains **O(LND)**.
- **Weight-Tied Reversible GNNs**:
    1. Combines weight-tying with **reversibility**.
    2. Memory is reduced to **O(ND)** (independent of depth).

Extending Weight-Tying to Groups

- Instead of weight-tying for all features, we divide features into **C groups**.
- Each group has its own tied function:

$$f_{w_i}^{(1)} := f_{w_i}^{(2)} \dots := f_{w_i}^{(L)}, \; i \in \{1, \cdots, C\} \qquad (10)$$

- Helps retain expressiveness while reducing parameters.

Weight-Tied Memory Complexity

Model Type	Memory Complexity
Regular Deep GNN	O(LND)
Weight-Tied Residual GNN	O(LND)
Weight-Tied Reversible GNN	O(ND)

Deep Equilibrium GNN: Overview

- Instead of explicitly stacking layers, the model **converges to a fixed point**.
- The equilibrium equation:

$$Z^* = f_w^{\text{DEQ}}(Z^*, X, A, U), \qquad (11)$$

- The network **iterates until it reaches this stable state**.
- No need to define a fixed depth for the model.

DEQ-GNN Equations

- The forward pass of DEQ-GNN is implemented with a root-finding algorithm (e.g. Broyden's method)
- The gradients are obtained by implicitly differentiating through the equilibrium node state for the backward pass.

$$Z' = \mathrm{GraphConv}(Z_{\mathrm{in}}, A, U) \tag{12}$$

$$Z'' = \mathrm{Norm}(Z' + X) \tag{13}$$

$$Z''' = \mathrm{GraphConv}(\mathrm{Dropout}(\mathrm{ReLU}(Z'')), A, U) \tag{14}$$

$$Z_{\mathrm{o}} = \mathrm{Norm}(\mathrm{ReLU}(Z''' + Z')), \tag{15}$$

UNIVERSITY OF WATERLOO

Memory Efficiency in DEQ-GNNs

Model Type	Memory Complexity
Regular Deep GNN	O(LND)
DEQ-GNN	O(ND)

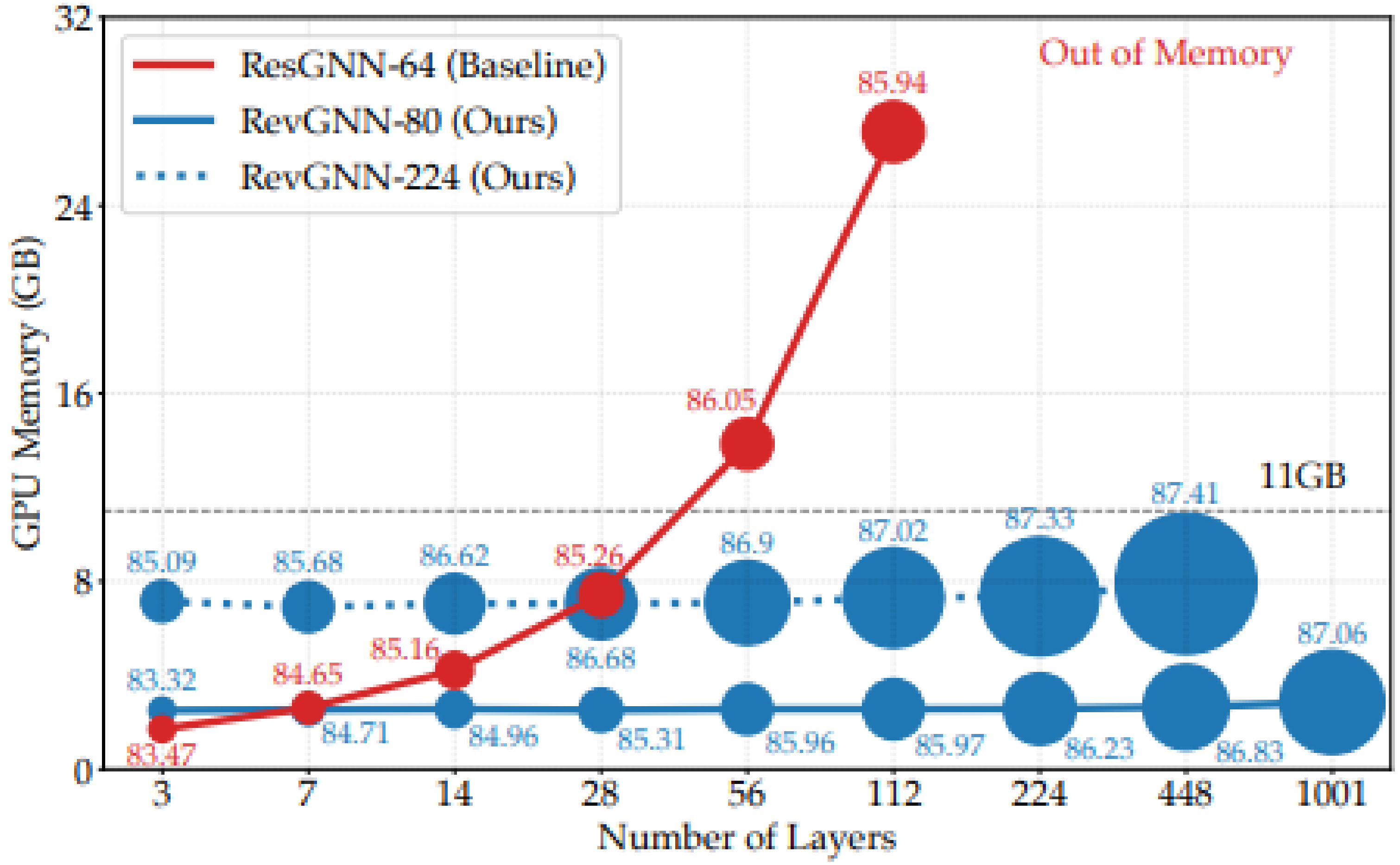
Analysis of Different Deep GNN Architectures

- **Dataset:** ogbn-proteins (OGB benchmark)
- **Training method:** Mini-batch training with random partitioning
- **Baseline Model:** ResGNN (Li et al., 2020)
- **Metric:** ROC-AUC score (higher is better)

Baseline GNN – ResGNN / Reversible GNN - RevGNN

- **Baseline** ResGNN
  - It’s a **pre-activation residual** GNN
- Reversible GNN - RevGNN
  - No need to store all activations
  - Constant memory usage
  - Scales beyond 1000 layers

ResGNN VS RevGNN

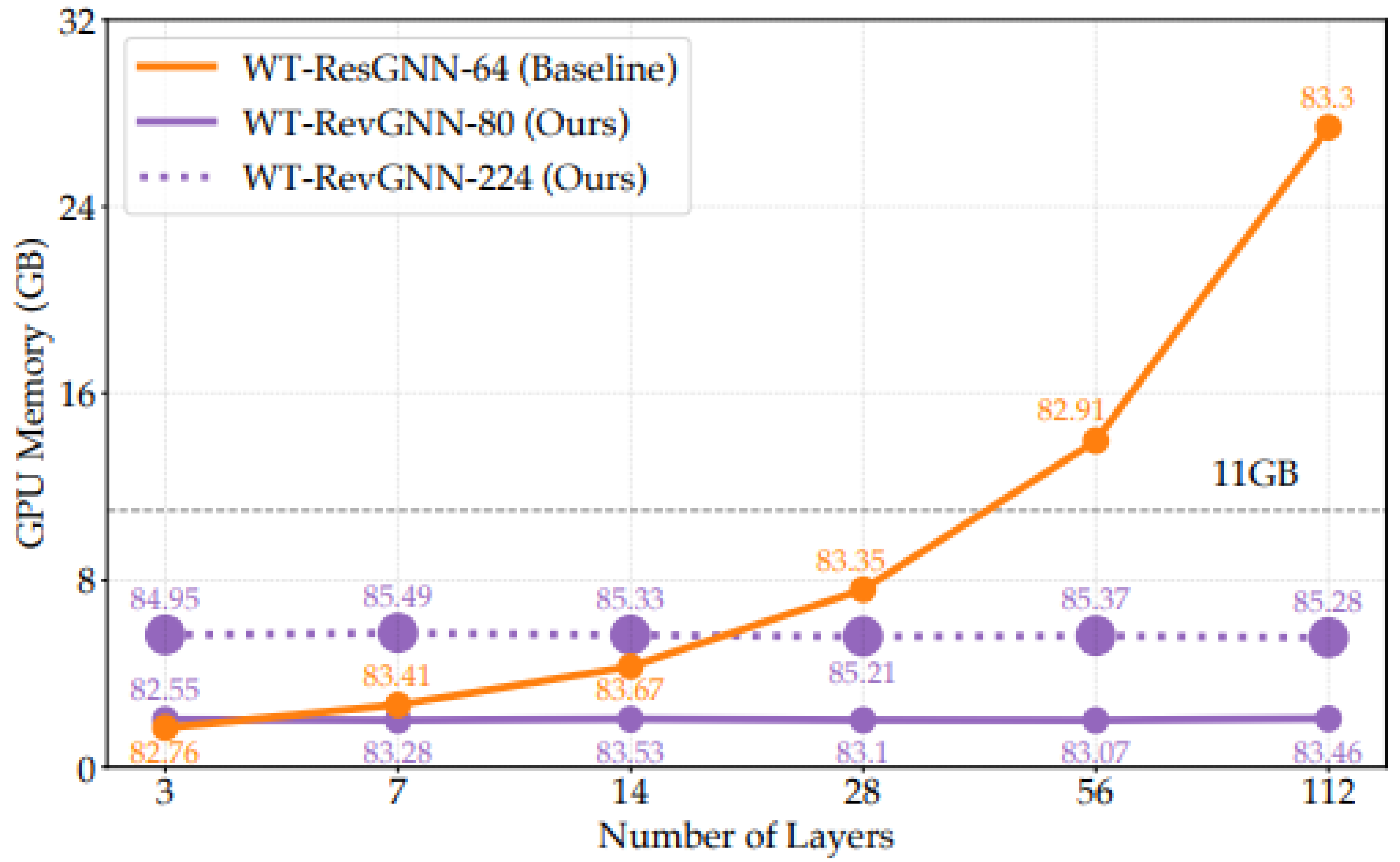


WT-ResGNN vs. WT-RevGNN

- **Sharing the same parameters** across all layers
- Parameter count **does not grow** with depth
- **WT-ResGNN**: Residual, weight-tied
- **WT-RevGNN**: Reversible, weight-tied
- Same number of parameters, **different memory usage**

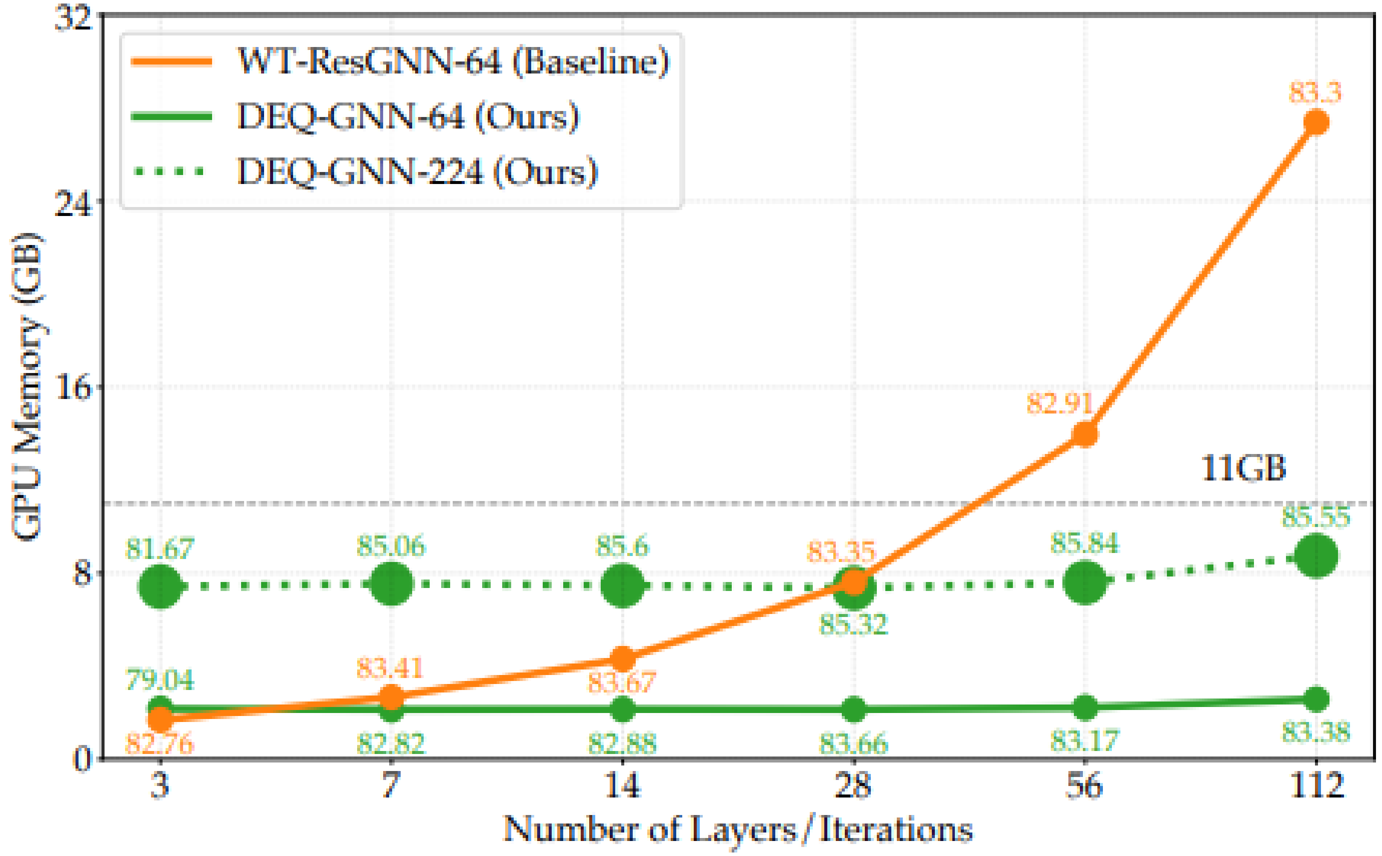
UNIVERSITY OF WATERLOO

WT-ResGNN vs. WT-RevGNN



UNIVERSITY OF WATERLOO

Equilibrium GNN



Over-parameterized Deep GNNs

- Experiments on **OGB** datasets (ogbn-proteins, ogbn-arxiv)
- Show **state-of-the-art** results with reversible GNN

ogbn-Protein Results

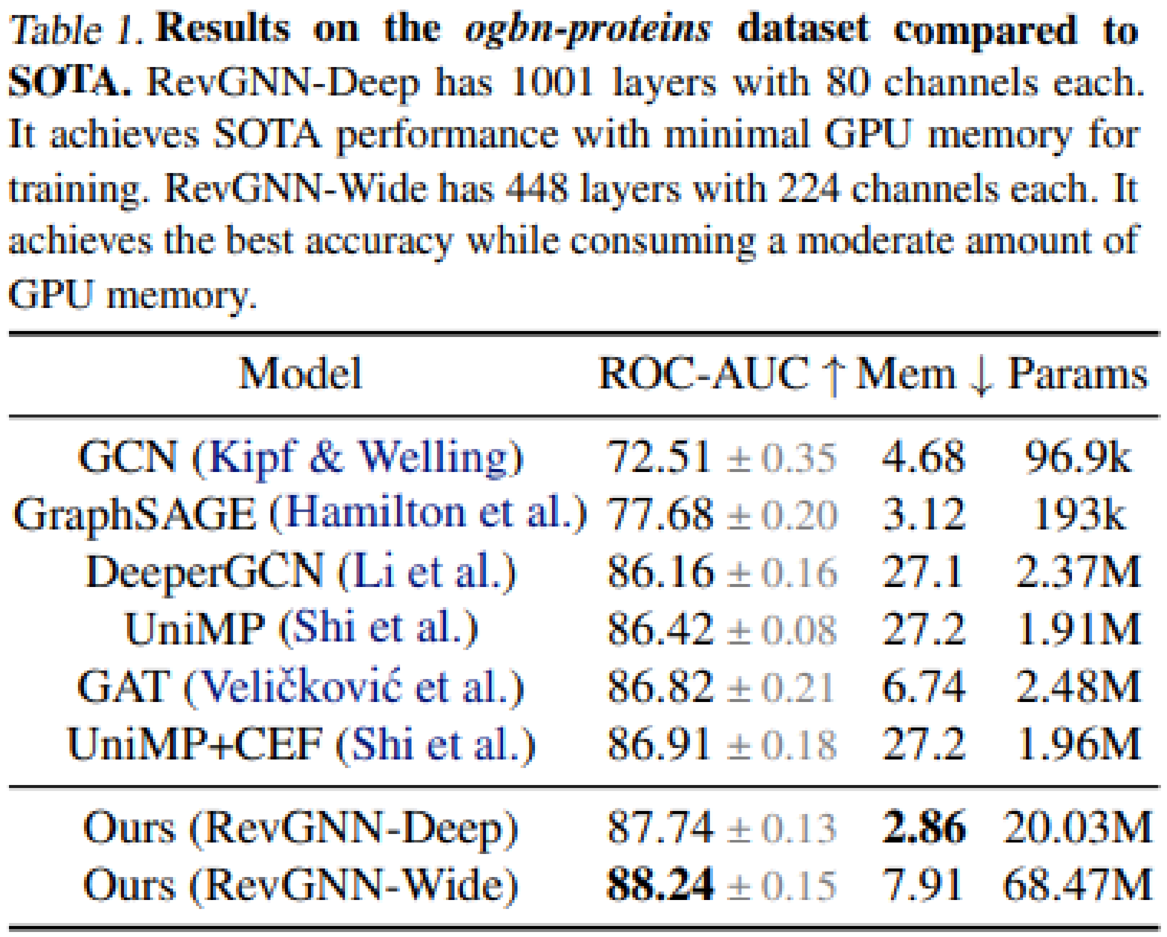


Table 1. **Results on the *ogbn-proteins* dataset compared to SOTA.** RevGNN-Deep has 1001 layers with 80 channels each. It achieves SOTA performance with minimal GPU memory for training. RevGNN-Wide has 448 layers with 224 channels each. It achieves the best accuracy while consuming a moderate amount of GPU memory.

Model	ROC-AUC ↑	Mem ↓	Params
GCN (Kipf & Welling)	72.51 ± 0.35	4.68	96.9k
GraphSAGE (Hamilton et al.)	77.68 ± 0.20	3.12	193k
DeeperGCN (Li et al.)	86.16 ± 0.16	27.1	2.37M
UniMP (Shi et al.)	86.42 ± 0.08	27.2	1.91M
GAT (Veličković et al.)	86.82 ± 0.21	6.74	2.48M
UniMP+CEF (Shi et al.)	86.91 ± 0.18	27.2	1.96M
Ours (RevGNN-Deep)	87.74 ± 0.13	**2.86**	20.03M
Ours (RevGNN-Wide)	**88.24** ± 0.15	7.91	68.47M

ogbn-arxiv Results

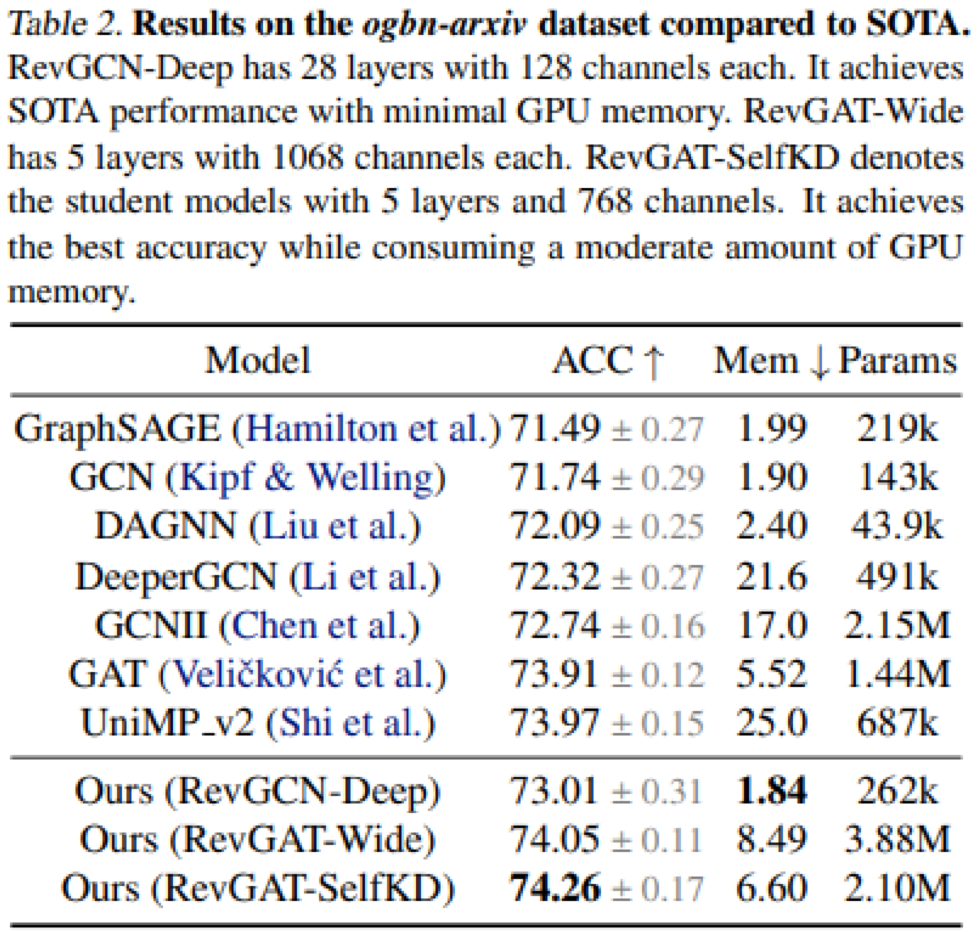


Table 2. **Results on the *ogbn-arxiv* dataset compared to SOTA.** RevGCN-Deep has 28 layers with 128 channels each. It achieves SOTA performance with minimal GPU memory. RevGAT-Wide has 5 layers with 1068 channels each. RevGAT-SelfKD denotes the student models with 5 layers and 768 channels. It achieves the best accuracy while consuming a moderate amount of GPU memory.

Model	ACC ↑	Mem ↓	Params
GraphSAGE (Hamilton et al.)	71.49 ± 0.27	1.99	219k
GCN (Kipf & Welling)	71.74 ± 0.29	1.90	143k
DAGNN (Liu et al.)	72.09 ± 0.25	2.40	43.9k
DeeperGCN (Li et al.)	72.32 ± 0.27	21.6	491k
GCNII (Chen et al.)	72.74 ± 0.16	17.0	2.15M
GAT (Veličković et al.)	73.91 ± 0.12	5.52	1.44M
UniMP_v2 (Shi et al.)	73.97 ± 0.15	25.0	687k
Ours (RevGCN-Deep)	73.01 ± 0.31	**1.84**	262k
Ours (RevGAT-Wide)	74.05 ± 0.11	8.49	3.88M
Ours (RevGAT-SelfKD)	**74.26** ± 0.17	6.60	2.10M

ogbn-arxiv Results on different GNN operators

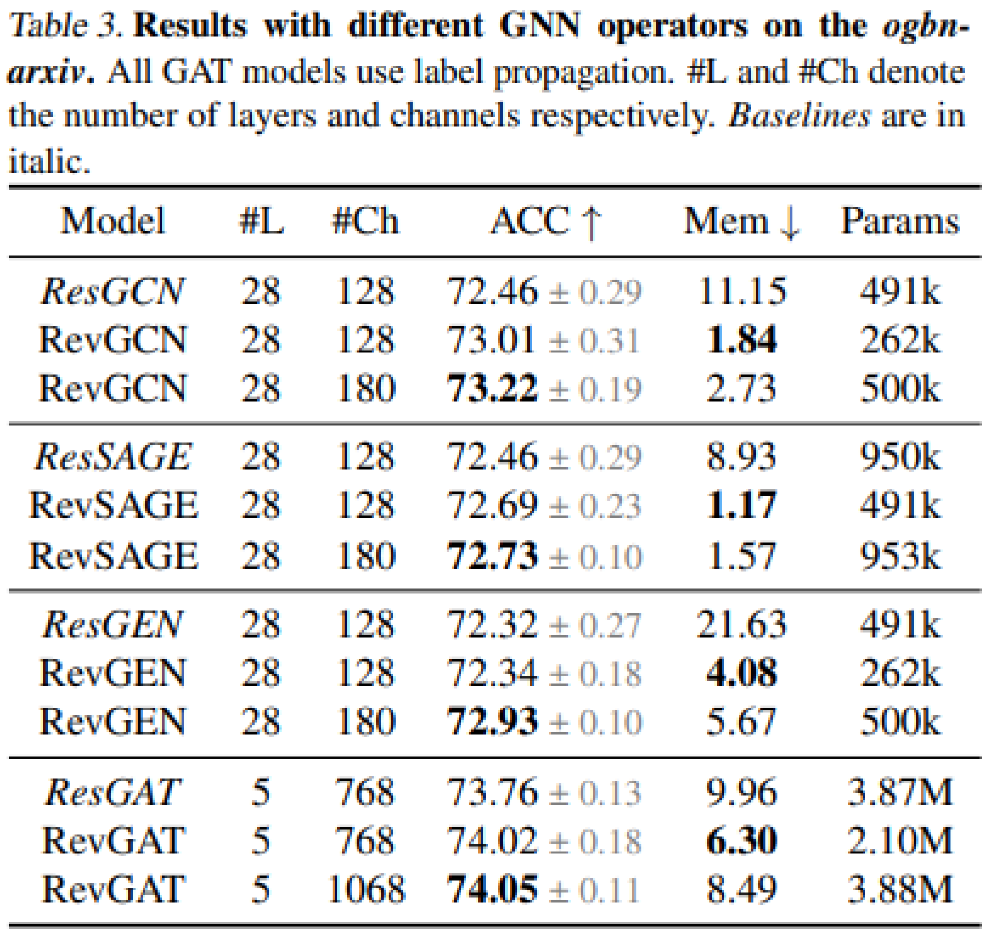


Table 3. **Results with different GNN operators on the *ogbn-arxiv*.** All GAT models use label propagation. #L and #Ch denote the number of layers and channels respectively. *Baselines* are in italic.

Model	#L	#Ch	ACC ↑	Mem ↓	Params
ResGCN	28	128	72.46 ± 0.29	11.15	491k
RevGCN	28	128	73.01 ± 0.31	**1.84**	262k
RevGCN	28	180	**73.22** ± 0.19	2.73	500k
ResSAGE	28	128	72.46 ± 0.29	8.93	950k
RevSAGE	28	128	72.69 ± 0.23	**1.17**	491k
RevSAGE	28	180	**72.73** ± 0.10	1.57	953k
ResGEN	28	128	72.32 ± 0.27	21.63	491k
RevGEN	28	128	72.34 ± 0.18	**4.08**	262k
RevGEN	28	180	**72.93** ± 0.10	5.67	500k
ResGAT	5	768	73.76 ± 0.13	9.96	3.87M
RevGAT	5	768	74.02 ± 0.18	**6.30**	2.10M
RevGAT	5	1068	**74.05** ± 0.11	8.49	3.88M

Analysis of Complexities

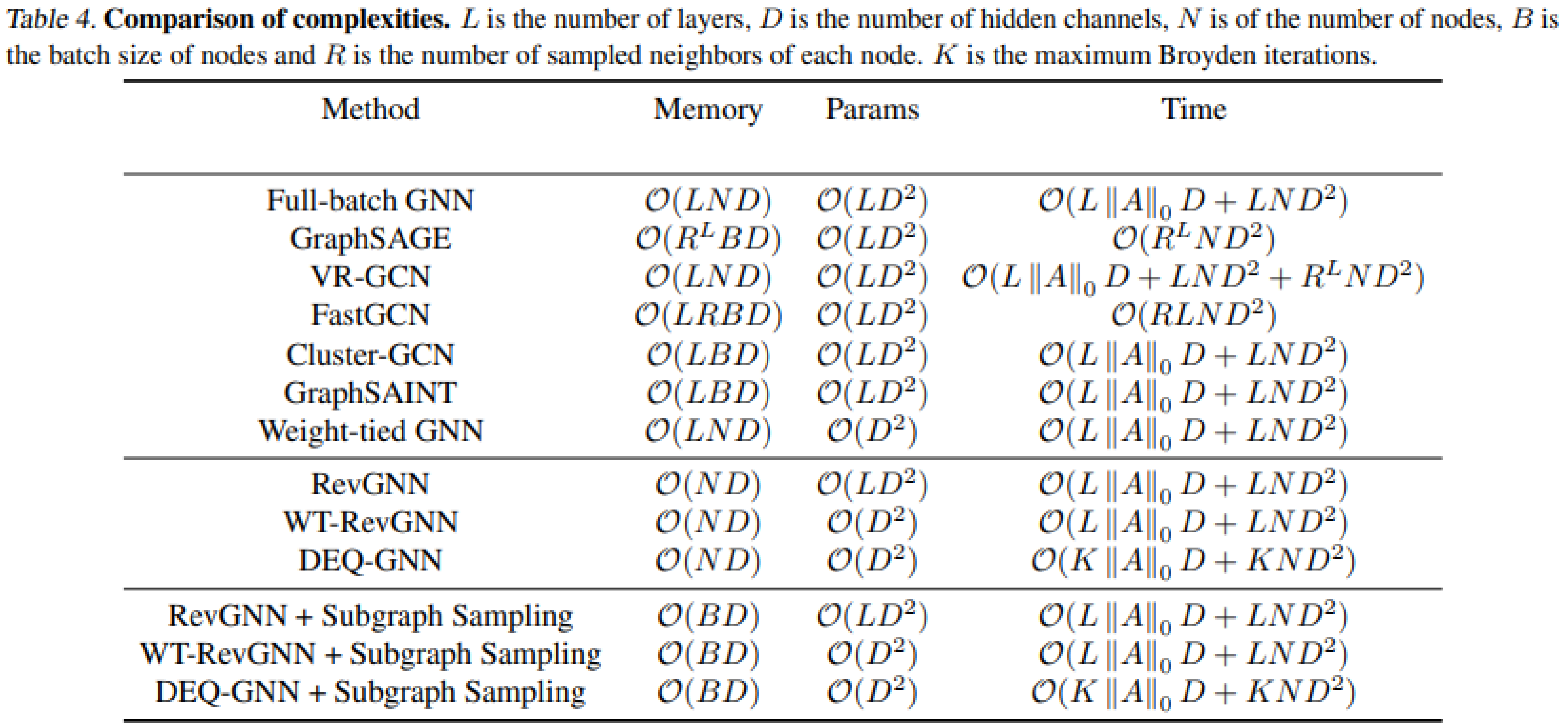
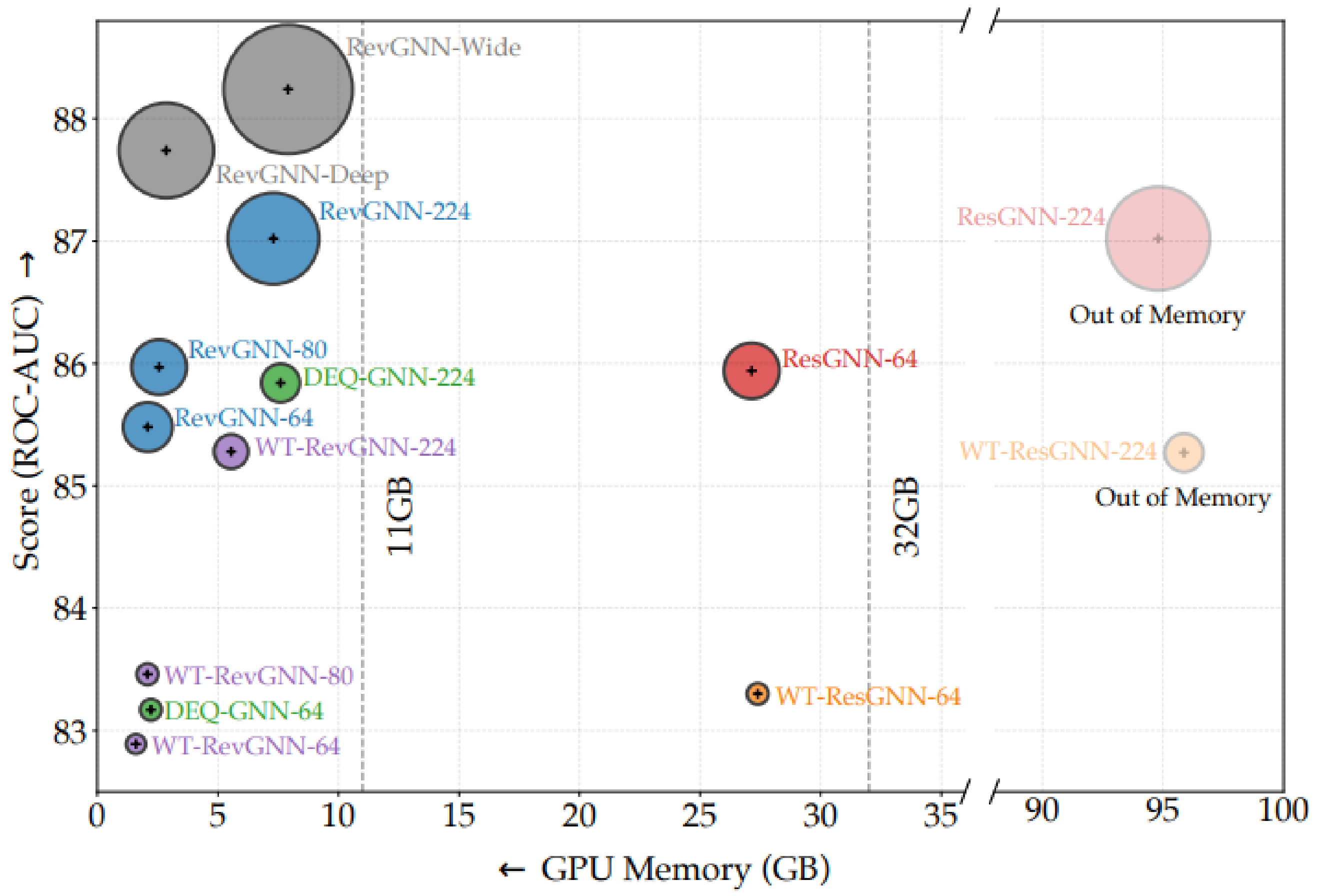


Table 4. **Comparison of complexities.** L is the number of layers, D is the number of hidden channels, N is of the number of nodes, B is the batch size of nodes and R is the number of sampled neighbors of each node. K is the maximum Broyden iterations.

Method	Memory	Params	Time
Full-batch GNN	$\mathcal{O}(LND)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
GraphSAGE	$\mathcal{O}(R^L BD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(R^L ND^2)$
VR-GCN	$\mathcal{O}(LND)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2 + R^L ND^2)$
FastGCN	$\mathcal{O}(LRBD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(RLND^2)$
Cluster-GCN	$\mathcal{O}(LBD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
GraphSAINT	$\mathcal{O}(LBD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
Weight-tied GNN	$\mathcal{O}(LND)$	$\mathcal{O}(D^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
RevGNN	$\mathcal{O}(ND)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
WT-RevGNN	$\mathcal{O}(ND)$	$\mathcal{O}(D^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
DEQ-GNN	$\mathcal{O}(ND)$	$\mathcal{O}(D^2)$	$\mathcal{O}(K\left\|A\right\|_0 D + KND^2)$
RevGNN + Subgraph Sampling	$\mathcal{O}(BD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
WT-RevGNN + Subgraph Sampling	$\mathcal{O}(BD)$	$\mathcal{O}(D^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
DEQ-GNN + Subgraph Sampling	$\mathcal{O}(BD)$	$\mathcal{O}(D^2)$	$\mathcal{O}(K\left\|A\right\|_0 D + KND^2)$

Results



UNIVERSITY OF WATERLOO

UNIVERSITY OF WATERLOO

Thank You!