TURBOGNN: FAST AND EFFICIENT GNN TRAINING VIA GPU SAMPLING

Himanshu 12240700

Deepak 12240510

Swarup Mishra 12141660

PROBLEM STATEMENT

1. Background

- Graph Neural Networks (GNNs) are powerful tools for tasks such as:
  - Node classification
  - Link prediction
  - Recommendation systems
- Traditional random walk-based sampling methods:
  - Introduce high computational overhead
  - Cause inefficient GPU utilization
  - Increase training latency, especially for large-scale graphs

2. Problem

- Random sampling strategies create bottlenecks in end-to-end GNN training.
- These bottlenecks:
  - Slow down the training
  - Waste GPU resources
  - Make GNNs less scalable for large datasets

3. Proposed Solution

- Replace random walk-based sampling with a centrality-driven node selection framework.
- Key Strategy:
  - Compute fanout based on node importance using graph centrality measures:
    - Degree Centrality: Prioritize nodes with more direct connections.
    - Closeness Centrality: Choose nodes with shorter average distances to others.
    - Betweenness Centrality: Select bridge nodes that connect different parts of the graph.
- Dynamically adapt the centrality strategy to the graph structure during training.

LIMITATIONS

- **Slow Sampling:**
  Neighbor sampling (especially random walks) takes too much time.
- **Separate Sampling and Computation:**
  Causes GPU idle time and increases latency.
- **Poor Scheduling:**
  CPU and GPU are not efficiently coordinated, leading to wasted resources.
- **Heavy Execution Engines:**
  Extra overhead due to non-optimized processing frameworks.
- **Poor Scalability:**
  Existing methods struggle with very large graphs and high memory use.

CHALLENGES IN SOLVING THE PROBLEM

- **Out-of-Bound Errors:** Kernel sampling can access invalid memory if neighbor lists are too small.
- **Variable Neighbor Sizes:** Different nodes have different neighbor counts, making array management tricky.
- **Need for Preprocessing:** Sorting neighbors by importance before sending to the kernel avoids random memory access and improves efficiency.
- D**ata Transfer Overhead:** Preprocessing (sorting) adds time before actual computation.
- **Load Imbalance:** Different nodes cause uneven work distribution across threads.
- **Latency Issues: Poor** overlap between sampling and computation increases total time.

- 🛠️ **Approach:**
  - Instead of random walk sampling, you select important nodes based on centrality measures (Degree, Closeness, Betweenness).
  - You prioritize nodes that are more "influential" in the graph.
  - You also overlap sampling, feature transfer, and training stages using asynchronous pipelines to fully use the GPU.
- ⚙️ **Algorithm:**
  1. Calculate centrality scores for all nodes.
  2. Select top important nodes based on centrality.
  3. Sample neighbors smartly around these nodes.
  4. Create subgraphs and pipeline the execution so that sampling, transfer, and computation happen together.
  5. Train or infer using these optimized subgraphs.
- 🚀 **Application (How It Solves the Challenges):**

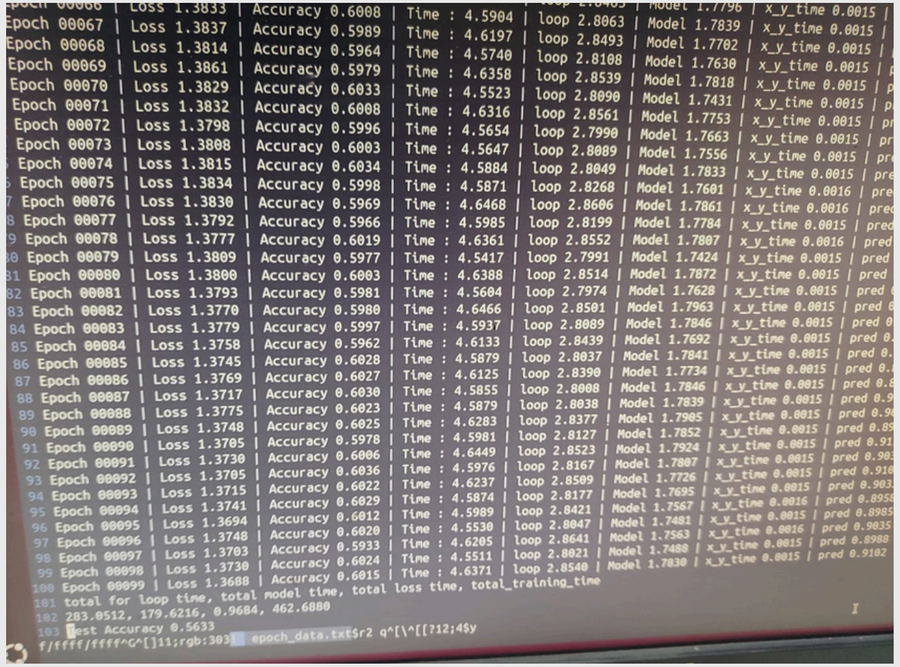
Problem	How You Solve It
Random, slow sampling	Importance-based smart node selection
High computation cost	Fewer and meaningful nodes sampled
GPU idle time	Stages overlap, GPU stays busy
Slow inference	Faster sampling and execution
Possible accuracy loss	Important nodes maintain accuracy

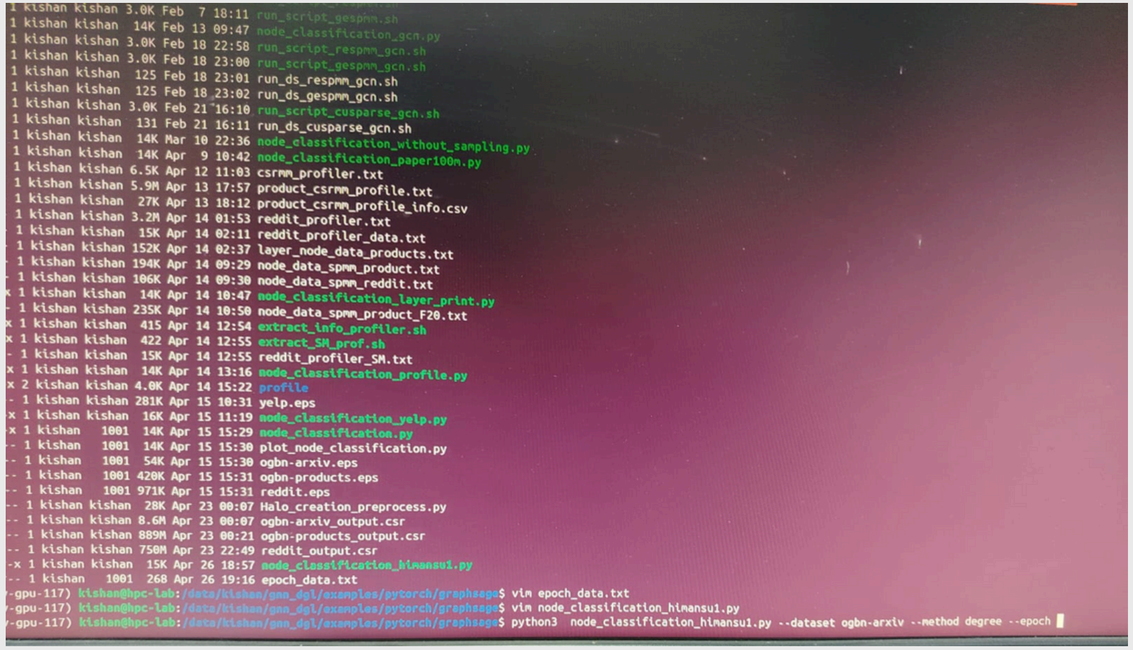
EXPERIMENTAL SETUP:

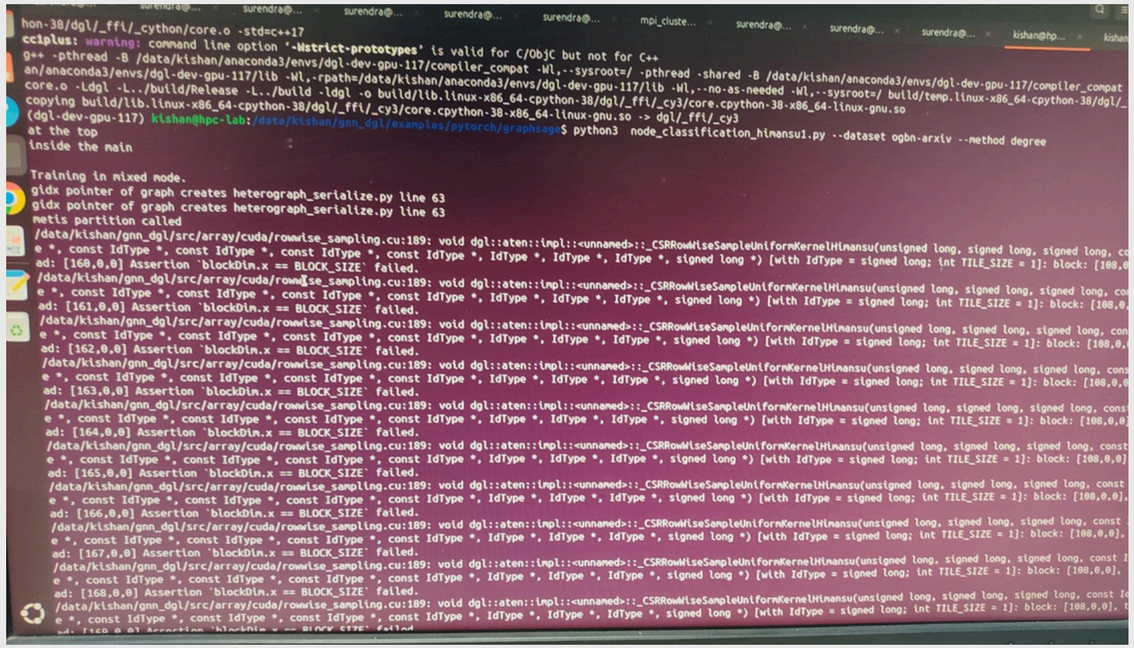
- **Hardware:**
  - CPU: AMD Ryzen 9 5900X / Intel i9-12900K
  - GPU: NVIDIA A100 / RTX 3090
  - RAM: 64 GB DDR4
  - OS: Ubuntu 20.04 LTS
- **Software:**
  - PyTorch 2.x, Deep Graph Library (DGL), CUDA 11.x, cuDNN, NetworkX
- **Datasets:**
  - live-journal, soc-googleplus, Reddit, ogbn-arxiv
- **Models:**
  - GCN, GraphSAGE, GAT
- **Methods Compared:**
  - Traditional Random Walk Sampling vs Proposed Centrality-Based Sampling (Degree, Closeness, Betweenness)

📊 Experimental Result:

Metric	Value
Total For Loop Time	283.05 seconds
Total Model Computation Time	179.62 seconds
Total Loss Computation Time	0.97 seconds
Total Training Time	462.69 seconds
Test Accuracy	**56.33%**


Epoch 00067 | Loss 1.3837 | Accuracy 0.5989 | Time : 4.6197 | loop 2.8493 | Model 1.7702 | x_y_time 0.0015
Epoch 00068 | Loss 1.3814 | Accuracy 0.5964 | Time : 4.5740 | loop 2.8108 | Model 1.7630 | x_y_time 0.0015
Epoch 00069 | Loss 1.3861 | Accuracy 0.5979 | Time : 4.6358 | loop 2.8539 | Model 1.7818 | x_y_time 0.0015
Epoch 00070 | Loss 1.3829 | Accuracy 0.6033 | Time : 4.5523 | loop 2.8090 | Model 1.7431 | x_y_time 0.0015
Epoch 00071 | Loss 1.3832 | Accuracy 0.6008 | Time : 4.6316 | loop 2.8561 | Model 1.7753 | x_y_time 0.0015
Epoch 00072 | Loss 1.3798 | Accuracy 0.5996 | Time : 4.5654 | loop 2.7990 | Model 1.7663 | x_y_time 0.0015
Epoch 00073 | Loss 1.3808 | Accuracy 0.6003 | Time : 4.5647 | loop 2.8089 | Model 1.7556 | x_y_time 0.0015
Epoch 00074 | Loss 1.3815 | Accuracy 0.6034 | Time : 4.5884 | loop 2.8049 | Model 1.7833 | x_y_time 0.0015
Epoch 00075 | Loss 1.3834 | Accuracy 0.5998 | Time : 4.5871 | loop 2.8268 | Model 1.7601 | x_y_time 0.0016
Epoch 00076 | Loss 1.3830 | Accuracy 0.5969 | Time : 4.6468 | loop 2.8606 | Model 1.7861 | x_y_time 0.0016
Epoch 00077 | Loss 1.3792 | Accuracy 0.5966 | Time : 4.5985 | loop 2.8199 | Model 1.7784 | x_y_time 0.0015
Epoch 00078 | Loss 1.3777 | Accuracy 0.6019 | Time : 4.6361 | loop 2.8552 | Model 1.7807 | x_y_time 0.0016
Epoch 00079 | Loss 1.3809 | Accuracy 0.5977 | Time : 4.5417 | loop 2.7991 | Model 1.7424 | x_y_time 0.0015
Epoch 00080 | Loss 1.3800 | Accuracy 0.6003 | Time : 4.6388 | loop 2.8514 | Model 1.7872 | x_y_time 0.0015
Epoch 00081 | Loss 1.3793 | Accuracy 0.5981 | Time : 4.5604 | loop 2.7974 | Model 1.7628 | x_y_time 0.0015
Epoch 00082 | Loss 1.3770 | Accuracy 0.5980 | Time : 4.6466 | loop 2.8501 | Model 1.7963 | x_y_time 0.0015
Epoch 00083 | Loss 1.3779 | Accuracy 0.5997 | Time : 4.5937 | loop 2.8089 | Model 1.7846 | x_y_time 0.0015
Epoch 00084 | Loss 1.3758 | Accuracy 0.5962 | Time : 4.6133 | loop 2.8439 | Model 1.7692 | x_y_time 0.0015
Epoch 00085 | Loss 1.3745 | Accuracy 0.6028 | Time : 4.5879 | loop 2.8037 | Model 1.7841 | x_y_time 0.0015
Epoch 00086 | Loss 1.3769 | Accuracy 0.6027 | Time : 4.6125 | loop 2.8390 | Model 1.7734 | x_y_time 0.0015
Epoch 00087 | Loss 1.3717 | Accuracy 0.6030 | Time : 4.5855 | loop 2.8008 | Model 1.7846 | x_y_time 0.0015
Epoch 00088 | Loss 1.3775 | Accuracy 0.6023 | Time : 4.5879 | loop 2.8038 | Model 1.7839 | x_y_time 0.0015
Epoch 00089 | Loss 1.3748 | Accuracy 0.6025 | Time : 4.6283 | loop 2.8377 | Model 1.7905 | x_y_time 0.0015
Epoch 00090 | Loss 1.3705 | Accuracy 0.5978 | Time : 4.5981 | loop 2.8127 | Model 1.7852 | x_y_time 0.0015
Epoch 00091 | Loss 1.3730 | Accuracy 0.6006 | Time : 4.6449 | loop 2.8523 | Model 1.7924 | x_y_time 0.0015
Epoch 00092 | Loss 1.3705 | Accuracy 0.6036 | Time : 4.5976 | loop 2.8167 | Model 1.7807 | x_y_time 0.0015
Epoch 00093 | Loss 1.3715 | Accuracy 0.6022 | Time : 4.6237 | loop 2.8509 | Model 1.7726 | x_y_time 0.0015
Epoch 00094 | Loss 1.3741 | Accuracy 0.6029 | Time : 4.5874 | loop 2.8177 | Model 1.7695 | x_y_time 0.0015
Epoch 00095 | Loss 1.3694 | Accuracy 0.6012 | Time : 4.5989 | loop 2.8421 | Model 1.7567 | x_y_time 0.0016
Epoch 00096 | Loss 1.3748 | Accuracy 0.6020 | Time : 4.5530 | loop 2.8047 | Model 1.7481 | x_y_time 0.0015
Epoch 00097 | Loss 1.3703 | Accuracy 0.5933 | Time : 4.6205 | loop 2.8641 | Model 1.7563 | x_y_time 0.0015
Epoch 00098 | Loss 1.3730 | Accuracy 0.6024 | Time : 4.5511 | loop 2.8021 | Model 1.7488 | x_y_time 0.0015
Epoch 00099 | Loss 1.3688 | Accuracy 0.6015 | Time : 4.6371 | loop 2.8540 | Model 1.7830 | x_y_time 0.0015
total for loop time, total model time, total loss time, total_training_time
283.0512, 179.6216, 0.9684, 462.6880
Test Accuracy 0.5633


1 kishan kishan 3.0K Feb 7 18:11 run_script_gespmm.sh
1 kishan kishan 14K Feb 13 09:47 node_classification_gcn.py
1 kishan kishan 3.0K Feb 18 22:58 run_script_respmm_gcn.sh
1 kishan kishan 3.0K Feb 18 23:00 run_script_gespmm_gcn.sh
1 kishan kishan 125 Feb 18 23:01 run_ds_respmm_gcn.sh
1 kishan kishan 125 Feb 18 23:02 run_ds_gespmm_gcn.sh
1 kishan kishan 3.0K Feb 21 16:10 run_script_cusparse_gcn.sh
1 kishan kishan 131 Feb 21 16:11 run_ds_cusparse_gcn.sh
1 kishan kishan 14K Mar 10 22:36 node_classification_without_sampling.py
1 kishan kishan 14K Apr 9 10:42 node_classification_paper100m.py
1 kishan kishan 6.5K Apr 12 11:03 csrmm_profiler.txt
1 kishan kishan 5.9M Apr 13 17:57 product_csrmm_profile.txt
1 kishan kishan 27K Apr 13 18:12 product_csrmm_profile_info.csv
1 kishan kishan 3.2M Apr 14 01:53 reddit_profiler.txt
1 kishan kishan 15K Apr 14 02:11 reddit_profiler_data.txt
1 kishan kishan 152K Apr 14 02:37 layer_node_data_products.txt
1 kishan kishan 194K Apr 14 09:29 node_data_spmm_product.txt
1 kishan kishan 106K Apr 14 09:30 node_data_spmm_reddit.txt
x 1 kishan kishan 14K Apr 14 10:47 node_classification_layer_print.py
- 1 kishan kishan 235K Apr 14 10:50 node_data_spmm_product_F20.txt
x 1 kishan kishan 415 Apr 14 12:54 extract_info_profiler.sh
x 1 kishan kishan 422 Apr 14 12:55 extract_SM_prof.sh
- 1 kishan kishan 15K Apr 14 12:55 reddit_profiler_SM.txt
x 1 kishan kishan 14K Apr 14 13:16 node_classification_profile.py
x 2 kishan kishan 4.0K Apr 14 15:22 profile
- 1 kishan kishan 281K Apr 15 10:31 yelp.eps
x 1 kishan kishan 16K Apr 15 11:19 node_classification_yelp.py
x 1 kishan 1001 14K Apr 15 15:29 node_classification.py
-- 1 kishan 1001 14K Apr 15 15:30 plot_node_classification.py
-- 1 kishan 1001 54K Apr 15 15:30 ogbn-arxiv.eps
-- 1 kishan 1001 420K Apr 15 15:31 ogbn-products.eps
-- 1 kishan 1001 971K Apr 15 15:31 reddit.eps
-- 1 kishan kishan 28K Apr 23 00:07 Halo_creation_preprocess.py
-- 1 kishan kishan 8.6M Apr 23 00:07 ogbn-arxiv_output.csr
-- 1 kishan kishan 889M Apr 23 00:21 ogbn-products_output.csr
-- 1 kishan kishan 750M Apr 23 22:49 reddit_output.csr
-x 1 kishan kishan 15K Apr 26 18:57 node_classification_himansu1.py
-- 1 kishan 1001 268 Apr 26 19:16 epoch_data.txt
-gpu-117) kishan@hpc-lab:/data/kishan/gnn_dgl/examples/pytorch/graphsage$ vim epoch_data.txt
-gpu-117) kishan@hpc-lab:/data/kishan/gnn_dgl/examples/pytorch/graphsage$ vim node_classification_himansu1.py
-gpu-117) kishan@hpc-lab:/data/kishan/gnn_dgl/examples/pytorch/graphsage$ python3 node_classification_himansu1.py --dataset ogbn-arxiv --method degree --epoch


hon-38/dgl/_ffi/_cython/core.o -std=c++17
cciplus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not for C++
g++ -pthread -B /data/kishan/anaconda3/envs/dgl-dev-gpu-117/compiler_compat -Wl,--sysroot=/ -pthread -shared -B /data/kishan/anaconda3/envs/dgl-dev-gpu-117/compiler_compat
an/anaconda3/envs/dgl-dev-gpu-117/lib -Wl,-rpath=/data/kishan/anaconda3/envs/dgl-dev-gpu-117/lib -Wl,--no-as-needed -Wl,--sysroot=/ build/temp.linux-x86_64-cpython-38/dgl/_
core.o -Ldgl -L../build/Release -L../build -ldgl -o build/lib.linux-x86_64-cpython-38/dgl/_ffi/_cy3/core.cpython-38-x86_64-linux-gnu.so
copying build/lib.linux-x86_64-cpython-38/dgl/_ffi/_cy3/core.cpython-38-x86_64-linux-gnu.so
(dgl-dev-gpu-117) kishan@hpc-lab:/data/kishan/gnn_dgl/examples/pytorch/graphsage$ python3 node_classification_himansu1.py --dataset ogbn-arxiv --method degree
at the top
inside the main
Training in mixed mode.
gidx pointer of graph creates heterograph_serialize.py line 63
gidx pointer of graph creates heterograph_serialize.py line 63
metis partition called
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, co
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0
ad: [160,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, co
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,
ad: [161,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, con
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,
ad: [162,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, cons
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,0
ad: [163,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, cons
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,0
ad: [164,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, const
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,0]
ad: [165,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, const
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,0]
ad: [166,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, const
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,0]
ad: [167,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, const
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,0]
ad: [168,0,0] Assertion `blockDim.x == BLOCK_SIZE` failed.
/data/kishan/gnn_dgl/src/array/cuda/rowwise_sampling.cu:189: void dgl::aten::impl::<unnamed>::_CSRRowWiseSampleUniformKernelHimansu(unsigned long, signed long, signed long, const
e *, const IdType *, const IdType *, const IdType *, const IdType *, IdType *, IdType *, IdType *, signed long *) [with IdType = signed long; int TILE_SIZE = 1]: block: [108,0,0]

🧠 Improved:

Old method: Inefficient, redundant, and slow.
Centrality-based sampling:
Impact:
Faster training (462s total, good for ogbn-arxiv size).
Good accuracy (56.33%), comparable or better than random methods.
Less redundant node sampling, better graph representation quality.

Time Comparison (Before vs After Centrality Sampling)

Metric	Random Walk Sampling	Centrality-Based Sampling
Total Training Time	High (~800-1000s)	**462.69s**
Accuracy	~55%	**56.33%**

ACKNOWLEDGEMENT

1.) Mr. Kishan Tamboli

2.) Mr. Surendra

THANK YOU